



## 저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Ph.D. DISSERTATION

# Object Detection and Classification in 3D Point Cloud Data for Automated Driving

자율 주행을 위한 3D Point Cloud Data 기반 물체 탐지 및  
분류 기법에 관한 연구

Myung-Ok Shin

FEBRUARY 2017

DEPARTMENT OF ELECTRICAL ENGINEERING AND  
COMPUTER SCIENCE  
GRADUATE SCHOOL  
SEOUL NATIONAL UNIVERSITY



# Abstract

A 3D LIDAR provides 3D surface information of objects with the highest position accuracy, among available sensors that can be utilized to develop perception algorithms for automated driving vehicles. In terms of automated driving, the accurate surface information gives the following benefits: 1) the accurate position information that is quite useful itself for collision avoidance is stably provided regardless of illumination condition, because the LIDAR is an active sensor. 2) the surface information can provide precise 3D shape-oriented features for object classification. Motivated by these characteristics, we propose three algorithms for a perception purpose of automated driving vehicles based on the 3D LIDAR in this dissertation.

A very first procedure to utilize the 3D LIDAR as a perception sensor is segmentation that transform a stream of the LIDAR measurements into multiple point groups, where each point group indicate an individual object near the sensor. In chapter 2, a real-time and accurate segmentation is proposed. In particular, Gaussian Process regression is used to solve a problem called over-segmentation that increases False Positives by partitioning an object into multiple portions.

The segmentation result can be utilized as input of another perception algorithm, such as object classification that is required for designing more human-likely driving strategies. For example, it is important to recognize pedestrians in urban driving environments because avoiding collisions with pedestrians are nearly a top priority. In chapter 3, we propose a pedestrian recognition algorithm based on a

Deep Neural Network architecture that learns appearance variation.

Another traffic participant that should be recognized with high-priority is a vehicle. Because various vehicle types of which appearances differ, such as a sedan, a bus, or a truck, are present on road, detection of the vehicles with similar performance regardless of the types is necessary. In chapter 4, we propose an algorithm that makes use of a common appearance of vehicles to solve the problem. To improve performance, a monocular camera is additionally employed, where the information from both sensors are integrated by a Dempster-Shafer Theory framework.

**Keywords:** 3D LIDAR, Real-time Segmentation, Gaussian Process, Pedestrian Recognition, Deep Neural Network, Vehicle Recognition

**Student Number:** 2010-20826

# Contents

<b>Abstract</b>	<b>i</b>
<b>Chapter 1 Introduction</b>	<b>1</b>
1.1 Background and Motivations . . . . .	1
1.2 Contributions and Outline of the Dissertation . . . . .	3
1.2.1 Real-time and Accurate Segmentation of 3D Point Clouds based on Gaussian Process Regression . . . . .	3
1.2.2 Pedestrian Recognition Based on Appearance Variation Learn- ing . . . . .	4
1.2.3 Vehicle Recognition using a Common Appearance Captured by a 3D LIDAR and a Monocular Camera . . . . .	5
<b>Chapter 2 Real-time and Accurate Segmentation of 3D Point Clouds                 based on Gaussian Process Regression</b>	<b>6</b>
2.1 Introduction . . . . .	6
2.2 Related Work . . . . .	10
2.3 Framework overview . . . . .	15

2.4	Clustering of Non-ground Points . . . . .	16
2.4.1	Graph Construction . . . . .	17
2.4.2	Clustering of Points on Vertical Surface . . . . .	17
2.4.3	Cluster Extension . . . . .	21
2.5	Accuracy Enhancement . . . . .	24
2.5.1	Approach to Handling Over-segmentation . . . . .	26
2.5.2	Handling Over-segmentation with GP Regression . . . . .	27
2.5.3	Learning Hyperparameters . . . . .	31
2.6	Experiments . . . . .	32
2.6.1	Experiment Environment . . . . .	32
2.6.2	Evaluation Metrics . . . . .	33
2.6.3	Processing Time . . . . .	36
2.6.4	Accuracy on Various Driving Environments . . . . .	37
2.6.5	Impact on Tracking . . . . .	46
2.7	Conclusion . . . . .	48

## **Chapter 3 Pedestrian recognition based on appearance variation**

	<b>learning</b>	<b>50</b>
3.1	Introduction . . . . .	50
3.2	Related Work . . . . .	53
3.3	Appearance Variation Learning . . . . .	56
3.3.1	Primal Input Data for the Proposed Architecture . . . . .	57
3.3.2	Learning Spatial Features from Appearance . . . . .	57
3.3.3	Learning Appearance Variation . . . . .	59

3.3.4	Classification . . . . .	61
3.3.5	Data Augmentation . . . . .	61
3.3.6	Implementation Detail . . . . .	61
3.4	EXPERIMENTS . . . . .	62
3.4.1	Experimental Environment . . . . .	62
3.4.2	Experimental Results . . . . .	65
3.5	CONCLUSIONS AND FUTURE WORKS . . . . .	70
 <b>Chapter 4 Vehicle Recognition using a Common Appearance Cap-</b>		
	<b>tured by a 3D LIDAR and a Monocular Camera</b>	<b>72</b>
4.1	Introduction . . . . .	72
4.2	Related Work . . . . .	75
4.3	Vehicle Recognition . . . . .	77
4.3.1	Point Cloud Processing . . . . .	78
4.3.2	Image Processing . . . . .	80
4.3.3	Dempster-Shafer Theory (DST) for Information Fusion . . .	82
4.4	Experiments . . . . .	84
4.5	Conclusion . . . . .	87
 <b>Chapter 5 Conclusion</b>		<b>89</b>



# List of Figures

Figure 2.1	Example of an image in which two vehicles suffer from over-segmentation. The two vehicles are separated into upper and lower parts shown in red and green, respectively. Light gray points indicate the other measurement points from a 3D LIDAR. . . . .	8
Figure 2.2	Processing flow of the proposed segmentation algorithm. . .	16
Figure 2.3	Clustering of points on vertical surface. (a) In each layer, line segments are generated by clustering $p_{u,v} \in C$ toward horizontal direction. (b) Neighboring line segments are merged if a distance between their end points is shorter than $\tau_d$ . . .	21
Figure 2.4	Cluster extension. (a) Lower boundary between an object and ground is detected by confirming slopes of vertical neighbor nodes. (b) After the lower boundaries are determined, remaining $p_{u,v} \in C$ are merged into the nearest objects based on Euclidean-distance. . . . .	23

Figure 2.5	Entire process of over-segmentation handling. (a) Example of over-segmentation, shown in orange and light green. (b) Overlapping areas of neighboring objects in the $x - y$ plane. (c) Test points (red) and training points (blue) of the paired objects $P_{e(t_1)}$ (orange) and $P_{e(t_2)}$ (light green). (d) Example of GP regression in a Cartesian coordinate. $z$ coordinates of the light blue points are $\bar{\mathbf{f}}_*$ approximated by Eq. (2.19) and $z$ coordinates of the red points are $\mathbf{f}_{actual}$ . (e) Result of over-segmentation handling. Only portions of the vehicle are merged, although a couple of adjacent objects were overlapped in (b). . . . .	25
Figure 2.6	Experiment environment. (a) Campus road of SNU. (b) Pitch angle variation with the frame number. . . . .	33
Figure 2.7	Example of the <i>target objects</i> , colored green, in a single frame.	34
Figure 2.8	Comparison of the processing times for 5938 frames from our own data set on an Intel Core i7-4770K at 3.5 GHz. . .	37
Figure 2.9	Comparisons of over-segmentation handling results. A bus, an SUV, and a sedan are represented in each row from the left. (a) Our algorithm. (b) Mesh-based algorithm [1]. (c) Polar grid-based algorithm [2]. . . . .	40
Figure 2.10	Segmentation results of the proposed algorithm. Most <i>target objects</i> are (a) pedestrians, (b) vehicles, and (c) pedestrians and vehicles at an intersection. Each color represents an object separated, while dark yellow indicates the ground. . .	40

Figure 2.11	Illustration of ground extraction failure. (a)-(b) and (c)-(d) are images of segmentation results of the same frame, respectively. In contrast to our algorithm in (a) and (c), the pink and the light green areas in (b) and (d) are totally distorted due to the faulty ground extraction. (b) is the result of the polar grid-based algorithm [2] and (d) is the result of the mesh-based algorithm [1]. . . . .	42
Figure 3.1	Point clouds in driving environments. (a) Point density variation of pedestrians. Distances of the left and the right are approximately 21 m and 40 m, respectively. (b) Similar appearances of a pedestrian (left) and a pole (right), both at a distance of approximately 40 m. . . . .	51
Figure 3.2	The overall architecture of the proposed DNN. . . . .	56
Figure 3.3	Sequential point clouds. Green indicates the earliest observation in $t_{-3}$ and red indicates the current observation in $t_0$ . (a) Pedestrian (b) Non-pedestrian . . . . .	58
Figure 3.4	Incorrect data association. (a) Pedestrian (b) Non-pedestrian	58
Figure 3.5	Recalls of the compared architectures according to distance	66
Figure 3.6	(a) A point cloud for a pedestrian and the corresponding voxel grid. (b) Weights of four 3D CNN kernels. Darker cells indicate higher values. (c) Activation of the first CNN layer computed by the four kernels in (b). . . . .	68

Figure 3.7	(a) Raw data sequence (b) LSTM activations (c) LSTM activation comparison according to time interval . . . . .	69
Figure 4.1	Different appearances of the vehicles. . . . .	73
Figure 4.2	Processing flow. . . . .	78
Figure 4.3	Feature extraction by shape fitting from point clouds. (a) "L" shape fitting on a sedan. The angle of the two red lines is extracted. (b) "I" shape fitting on a bus. The angle of the red line is extracted. The green points are points near the <i>BumperHeight</i> in both (a) and (b). . . . .	79
Figure 4.4	Feature extraction from a camera image. (a) A vehicle and a sliding window on the tires and a bumper. (b) "n" shape of low intensity distribution (top) and CCL result on the sliding window image (bottom). (c) A Canny edge map (top) and the extracted directional lines. . . . .	80
Figure 4.5	Comparison of the vehicle recognition results. (a) The result of the point cloud processing with the line feature extraction. (b) The result of the image processing obtained by a sliding window operation over the entire image. (c) The result of the information fusion by the DST framework. . . .	86
Figure 4.6	Example of FPs. ROIs where the directional lines are observed due to the contour between adjacent scene mostly generated FPs. . . . .	87

# List of Tables

Table 2.1	Comparison with the representative previous works. . . . .	15
Table 2.2	Processing time on our own dataset. . . . .	37
Table 2.3	Parameter setting. . . . .	38
Table 2.4	Accuracy comparison on our own dataset. . . . .	38
Table 2.5	Contextual performance variation on our own dataset. . . . .	41
Table 2.6	Accuracy comparison on the KITTI dataset. . . . .	44
Table 2.7	Impact on tracking. . . . .	47
Table 3.1	Positive sample composition ratio according to distance . . . .	63
Table 3.2	Performance comparison . . . . .	66
Table 3.3	Recalls of the compared algorithms according to distance . . .	67
Table 4.1	Quantitative evaluation. . . . .	88

# Chapter 1

## Introduction

### 1.1 Background and Motivations

Automated driving [3–5] is becoming a familiar concept. Technologies related to the automated driving, such as lane keeping or smart cruise control, having been spread to the public, particularly in highway driving conditions. As a result, safety and efficiency [6] of the practical driving are expected to be improved gradually.

However, for the ideal automated driving that can achieve a door to door driving, various element technologies, such as map building [7], localization [8], perception [9, 10], and decision making [11], should be developed with higher accuracy and reliability. Among them, the perception technology that recognizes surrounding objects is of great importance in that the technology functions as eyes of a person.

A 3D LIDAR is quite distinguishable from the other types of perception sen-

sors, with respect to the sensing mechanism. As an active sensor, the LIDAR emits multiple lasers and measures the time of flight of the lasers reflected from objects. Thus, the sensing capability is not affected by varying illumination, which is contrary to a camera of which the sensing capability is quite affected by the varying illumination, due to the passive sensing characteristic. In addition, while another type of a sensor, e.g., a radar, can only detect objects composed of metal, such as vehicles, the lasers are reflected nearly all types of object surfaces.

Because of the characteristics discussed above, information obtained by a 3D LIDAR are quite useful for the automated driving. First of all, the accuracy of the distance measurement is sufficiently high to avoid collisions in any illumination conditions. Next, 3D shape information can be employed for object classification, which cannot be obtained based on a single 2D LIDAR [10]. However, to make use of point clouds acquired by the 3D LIDAR for the aforementioned purposes, the development of appropriate perception algorithms is necessary. Motivated by this fact, perception algorithms based on the 3D point cloud data are proposed in this dissertation.

Two categories of perception are mainly discussed in this dissertation, i.e., segmentation and classification. Segmentation is a very first procedure of LIDAR-based object detection that separates point clouds into multiple point groups, where each point group corresponds an individual object. Classification is a recognition process that categorizes an object into a specific class, such as pedestrians or vehicles. In chapter 2, a real-time and accurate segmentation algorithm is proposed. Particularly, over-segmentation problem that separates an object into multiple portions is solved based on Gaussian Process (GP) regression [12]. In chapter 3, a pedestrian

recognition algorithm based on appearance variation learning is proposed. In addition to spatial features of point clouds, temporal features that can be observed from pedestrians on road are learned by a Deep Neural Network (DNN) architecture employing 3D Convolutional Neural Network (CNN) layers [13] and a Long Short-Term Memory (LSTM) layer [14] in a sequential manner. In chapter 4, a vehicle recognition algorithm using a common appearance captured by a 3D LIDAR and a monocular camera is proposed, where the information obtained by different sensors are fused by a Dempster-Shafer Theory (DST) framework [15].

## **1.2 Contributions and Outline of the Dissertation**

### **1.2.1 Real-time and Accurate Segmentation of 3D Point Clouds based on Gaussian Process Regression**

In LIDAR-based object detection, accurate object segmentation is of great importance since segmentation is an essential preprocessing step for other perception tasks, such as classification and tracking. For segmenting objects, most of the previous methods have tried to eliminate the ground first, which typically incurs a considerable overhead in computation and inaccuracy in object detection with point clouds gathered by using 3D LIDARs. However, in many real-time applications, such as automated driving, segmentation should be performed within a specified time because even a small delay in computation could result in vehicle collisions. In chapter 2, we propose a real-time and accurate object segmentation algorithm for 3D point clouds which does not carry out ground extraction as a first step. In the proposed algorithm, we generate candidate points of objects and find their borders based on the integrated structure of a 2D grid and an undi-



rected graph, which enables fast processing and yields an accurate segmentation result independent of ground extraction error. In order to enhance segmentation accuracy, we employ Gaussian process which reduces over-segmentation that separates an object into multiple portions. We apply two types of Gaussian process models to alternately provide cues for merging adjacent over-segmented objects. Experimental results demonstrate that our work achieves a real-time processing speed and higher segmentation accuracy than previous works in most evaluation metrics. With the application to tracking, we show that the enhanced segmentation accuracy increases the tracking accuracy by 11.4% even in the worst case.

### **1.2.2 Pedestrian Recognition Based on Appearance Variation Learning**

DNNs are one of the most popular tools for developing object recognition algorithms, and due to their powerful performance, the tools are being employed in the automated driving research area based on various types of data, such as images and point clouds. However, in the case of point clouds, most previous research has utilized either 3D computer-aided design (CAD) data or noise-free and dense point cloud data for evaluation. In an actual driving environment, the latter are rarely obtained due to the motion of the ego-vehicle, occlusion, and/or point density variations according to distance. As a result, performance with previous algorithms cannot be guaranteed when they are applied to point clouds in driving environments. In chapter 3, we propose a DNN architecture to recognize pedestrians, which are one of the most frequently appearing objects in driving environments. To make full use of information obtained from point clouds in driving environments,

we employed 3D CNN layers and an LSTM layer in a sequential manner, which allows capture of the appearance variation of an object according to time interval. Experimental results using both KITTI dataset and our own dataset demonstrate that the proposed DNN architecture outperforms baseline algorithms in driving environments by appearance variation learning.

### **1.2.3 Vehicle Recognition using a Common Appearance Captured by a 3D LIDAR and a Monocular Camera**

A vehicle is another type of object that is frequently appearing in a driving environment. One challenging issue that should be considered on developing a vehicle recognition algorithm is an intra-class variation problem. For example, the performance of a learning algorithm trained by samples of only one type of vehicles, e.g., a sedan, degrades for the other types of vehicles, such as a bus, an SUV, or a truck, because their appearance actually differ. Collecting samples of all types of vehicles can be a solution, however, the task requires a large amount of time. As an alternative, utilizing features that can be commonly extracted from all types of vehicles can be considered. In chapter 4, a vehicle recognition algorithm of which a goal is to detect all types of vehicles by utilizing features extracted from tires and a bumper that are commonly observed on vehicles. Because tires and a bumper rarely can be captured based on a 3D LIDAR in a distant range, a monocular camera is employed as an additional sensor. To fuse information from both sensors, a DST framework is exploited. Through experiments, an improved recognition result by the information fusion is demonstrated and limits of the proposed approach are discussed.

## Chapter 2

# Real-time and Accurate Segmentation of 3D Point Clouds based on Gaussian Process Regression

### 2.1 Introduction

A 3D LIDAR is a popular sensor in environment perception research, as the sensor provides significantly more accurate 3D surface information of surrounding objects than other sensors, such as a camera or a radar. Segmentation is the first step to utilizing the 3D LIDAR as a perception sensor that separates raw data points so that the neighboring points on the same object surface can be grouped together. This step is of great importance because segmentation functions as object detection, and its quality directly affects the performances of other perception steps that operate based on segmentation, e.g. classification or tracking. Particularly, in the research of automated driving vehicles, where personal safety issues are interrelated,

the importance of segmentation is intensified.

An important factor that should be considered on developing a segmentation algorithm for automated driving vehicles is a real-time constraint. Even a small delay of a few milliseconds (ms) in the perception system would result in a retarded decision about vehicle motion, which can cause vehicle collisions. In the case of a 3D LIDAR providing millions of measurements per second, the design problem of the algorithm to handle the rich data in real-time is critical. However, in previous studies, time has been spent on possibly unnecessary computation, such as ground extraction [1, 2, 16, 17]. This is inefficient in terms of processing time and somewhat unstable because incorrect segmentation is brought about when errors occur in the ground extraction result. Particularly, the instability can degrade overall segmentation accuracy in an urban driving environment where varying slopes and bumpy roads frequently appear.

Another issue in segmentation is the phenomenon called over-segmentation, a form of faulty segmentation result. It disturbs the perception system of an automated driving vehicle by separating one object into multiple portions. For example, it divides an object, e.g. a vehicle, into more than two parts such as a bonnet, a bumper, or a roof as shown in Fig. 2.1; this causes the perception system to make the misjudgment that there are multiple small objects, rather than one object. The additional portions generated by over-segmentation not only reduce the precision of a segmentation algorithm, but also degrade the tracking algorithm. This problem cannot be handled with the clustering methods that are based on constant Euclidean-distances employed in previous researches [1, 2, 16–21] because the gaps between the over-segmented objects are not usually regular, particularly in urban



Figure 2.1 Example of an image in which two vehicles suffer from over-segmentation. The two vehicles are separated into upper and lower parts shown in red and green, respectively. Light gray points indicate the other measurement points from a 3D LIDAR.

driving environments.

In this chapter, we propose a real-time and accurate segmentation algorithm for 3D point clouds obtained by using a 3D LIDAR. As a first step, a 2D occupancy grid is utilized to determine the surrounding ground heights in a distributed manner, where a set of candidate points of non-ground objects higher than the ground heights are generated. To separate the non-ground objects, neighboring points in the set are then grouped based on an undirected graph, where the lower boundaries of the objects are found by confirming the slopes formed between the ground and the objects. By combining the grid and the graph structures, our algorithm skips ground extraction that was required in many previous studies, which contributes considerably to achieving a real-time performance of around 40.1 ms per frame on average. Also, our algorithm secures higher stability than the algorithms in previous works because our segmentation result is independent of the ground extraction error

in various road conditions.

We employ Gaussian Process (GP) regression [12] to improve the segmentation result by reducing the aforementioned over-segmentation. To decide whether two neighboring objects have been separated from the same object, GP regression approximates the surface coordinates of one object based on those of the other object. The two objects are then merged if the approximated values are similar to the actual values. Although additional under-segmentations whereby different objects are merged occur when over-segmentation handling fails, usually the number is considerably less than the number of over-segmentations successfully dealt with. This robustness results in higher segmentation accuracy than in the previous works and increases the performance of the tracking algorithm by 11.4%, even in the worst case.

The superiority of our algorithm is proved with a sufficiently large amount of data gathered in various driving environments. For quantitative analysis, we propose new metrics evaluating segmentation results in an object unit. The evaluation methods employed in previous studies [1, 22, 23] are inappropriate for our purpose because of the time-consuming task in those methods of labeling all the points of a frame, i.e. a point cloud of around 100,000 points from one LIDAR scan. In contrast, our metrics are based on *target objects* manually labeled in each frame, which enables the evaluation of 1,600 frames that is a considerably greater number of frames than that of 10 utilized in previous works.

The contributions of this chapter can be summarized as:

- We propose a method to obtain a real-time and stable segmentation result based on the combination of a 2D grid and an undirected graph structure.

- We propose a method of accuracy enhancement that reduces over-segmentation arising in the driving environment, which also improves tracking accuracy.
- We propose new evaluation metrics that are appropriate for testing a sufficiently large amount of data to verify the reliability of a segmentation algorithm in various urban driving environments.

The remainder of this chapter is organized as follows. In chapter 2.2, we present the related works. We provide a framework overview in chapter 2.3 and the detailed procedure of non-ground point clustering in chapter 2.4. Chapter 2.5 describes the accuracy enhancement that handles over-segmentation based on GP regression. We then provide experimental results in chapter 2.6 that verify the superiority of our algorithm. Finally, we conclude this chapter with a brief summary in chapter 2.7.

## 2.2 Related Work

The goal of our segmentation algorithm is to provide information about the surrounding objects including both dynamic objects and static objects for safe and efficient automated driving. Thus, a real-time constraint is of great importance in our study. A branch of research has focused on attempting to deeply understand static information, such as traffic signs, street lights, bus stations, and buildings [22, 24, 25]. In previous research, a large amount of time is typically spent utilizing multiple mathematical techniques, such as classification, Principal Component Analysis (PCA), and computation of normal vectors, to obtain precise road facility information. Also, previous studies utilize large-scale urban scene data accumulated by a mapping system, around hundreds of millions of points. Consequently,

the computation time amounts to many minutes, where the real-time constraint is a minor factor. In this section, this type of research is not introduced in detail because it is a significantly different area from that dealt with in this chapter.

A 2D rectangular grid plane is a well-known tool used to perform the segmentation of a 3D point cloud. The point cloud is projected onto an evenly divided 2D grid plane, and the maximum height differences of the points in the same cells are then computed. Cells of which the height differences exceed a predefined threshold are regarded as occupied. With the aid of the dimension reduction, this approach rapidly separates non-ground points from ground points, which is why it has been employed by many researchers [5, 18–21]. The non-ground points are then generally grouped using the connected component labeling (CCL) algorithm [18–21], well-known in the computer vision research area [26]. However, these types of approaches show an inherent limit of under-segmentation problem in which objects with different heights are merged together due to the dimensionality reduction in the same cell, e.g. a vehicle under branches of a tree. Also, some ground points are grouped with non-ground points in the same cell because the CCL is fulfilled on the 2D grid, which deforms the original shapes of the objects.

Some researchers have focused on performing ground extraction as a first step before clustering non-ground points. In their works, the extracted ground functions as a separator so that the neighboring non-ground points are grouped by simple clustering methods based on Euclidean-distance. The ground extraction differs from the type of a ground removal step based on the 2D grid in [5, 18–21] in that some mathematical techniques are required to label ground points and avoid the drawbacks of the ground removal that were discussed above. Douillard



*et al.* [1] proposed two different ground extraction methods, Gaussian Process Incremental SAmple Consensus (GP-INSAC) and a mesh-based segmentation, where non-ground points are clustered with voxel adjacency. Himmelsbach *et al.* exploited a local ground plane estimation algorithm with a 2D polar grid that was also able to handle sloped road, which was followed by CCL for non-ground point clustering in [2,16]. Chen *et al.* [23] combined the methods of [1] and [2] for ground extraction. They applied 1D GP regression to each 2D polar grid cell, which allowed finding ground faster than [1] with higher accuracy than [2]. The remaining non-ground points were separated into overhanging structures and obstacles on the road. Das *et al.* [27] mostly adopted the method of Chen *et al.* [23], except for non-ground points clustering that was performed by means of L2 norm. Reddy *et al.* [28,29] computed unevenness of each 3D point based on the range difference of consecutive points to determine ground points. Then, segmentation of the remaining non-ground points was performed using the region growing method, which successfully found small objects on the ground, such as road edges and speed breaks. Cheng *et al.* computed ring gradients to generate seed points indicating the vertical surface of non-ground objects [17]. Yin *et al.* [30] proposed a segmentation algorithm that only utilized distance and azimuth angle of measurement points computed in a spherical coordinate. Ground extraction is an essential process in this type of approach, which implies that the performance of the ground extraction affects the clustering result of non-ground points. As a result, in a case where ground extraction does not work well on some specific slope conditions, the overall segmentation performance can be degraded.

Segmentation algorithms have also been developed that neither depend on the

2D grid nor first extract the ground. Moosmann *et al.* [31] presented an algorithm using a local convexity criterion to group neighboring points, where the surface normals of points were computed. The authors found the surface normals relatively faster by calculating the cross product of the displacement vectors among 4-neighboring nodes, instead of least squares estimation. The combination of classification and Euclidean Minimum Spanning Tree - RANdom Sample Consensus (EMST-RANSAC) - has also been explored [32]. The authors performed patch segmentation that can suffer from over-segmentation, based on dissimilarity measures between adjacent points. Classification was applied to the patches to distinguish objects of interest, which was followed by EMST-RANSAC to deal with the over-segmentation.

These two algorithms [31, 32] could not meet the real-time constraint because the techniques employed were still heavy. For a case in which a target class is defined, an algorithm to detect only the target has also been developed [33]. Line segments in each 2D layer were generated and an adaboost was utilized to classify the line segments, where the classification results were combined to form a full person detector based on geometric relations. However, this type of approach is only valid for the target, where the same performance cannot be expected in a general environment that includes every type of traffic participant, such as vehicles, traffic signs, or trees.

To overcome the aforementioned problems, we combine a 2D grid and an undirected graph structure. In our work, the segmentation result is independent of the ground extraction [1, 2, 16, 17, 23, 28] error because we skip this procedure with the aid of the grid, which also enables our algorithm to achieve real-time performance.

Our algorithm carries out pointwise clustering with the graph structure, where the dimensionality reduction is non-essential. In addition, the proposed algorithm obtains higher segmentation accuracy than that in previous works by reducing over-segmentations. To the best of our knowledge, few studies have been presented that cope with the over-segmentation problem in real-time. Wang *et al.* [32] does not meet the real-time constraint, and Himmelsbach *et al.* [16] handles the over-segmentation problem with tracking information of consecutive frames, which could be further improved by enhancing the segmentation performance of a single frame, as shown in our work.

We employ GP regression [12], providing a powerful mathematical framework for surface estimation that is useful to handle over-segmentation. In the framework, the function values of test points are estimated with consideration of the sensor noise and correlation between training points. Also, any explicit mathematical models, such as linear, quadratic, or cubic, are unnecessary in GP regression; this has motivated many researchers to adopt GP framework in various applications. Interpolating gaps between 3D sensor data [34], estimating surface information of unexplored terrain [35], extracting ground [1, 23], detecting curbs [36], and approximating the behavior of vehicles or pedestrians [37, 38] are examples of such applications. In this chapter, two GP regression models are constructed, where each model is selected alternately according to the form of over-segmentation.

Table 2.1 shows a comparison of our algorithm with the two representative previous works, one of which has been most often cited in various works [1] and the other of which shows the fastest processing speed [2] among the related works mentioned so far. In chapter 2.6, we present a detailed performance comparison of

Table 2.1 Comparison with the representative previous works.

Algorithm	[1]	[2]	Ours
Robustness to under-segmentation	O	O	O
Independence from ground extraction	X	X	O
Robustness to over-segmentation	X	X	O
Real-time processing	X	O	O

experimental results.

## 2.3 Framework overview

Fig. 2.2 illustrates the processing flow of our segmentation algorithm, which carries out clustering of non-ground points based on Euclidean-distance as well as accuracy enhancement based on GP regression in a sequential manner.

As a preliminary step, we first construct an undirected graph for allowing fast access to unordered 3D measurement points, which particularly helps the clustering process. The surrounding ground heights are then determined on a 2D grid, where the lowest height of the points in each cell is measured. With the assumption that points higher than the ground heights are non-ground points, the non-ground points are clustered together based on Euclidean-distance to generate multiple chunks that indicate non-ground objects. The chunks are then extended until their lower boundaries that meet the ground are found by examining the slopes formed between vertical nodes.

Applying GP regression to each object, i.e. the chunk of 3D points, is time-consuming and prone to generate numerous under-segmentations. To achieve reli-

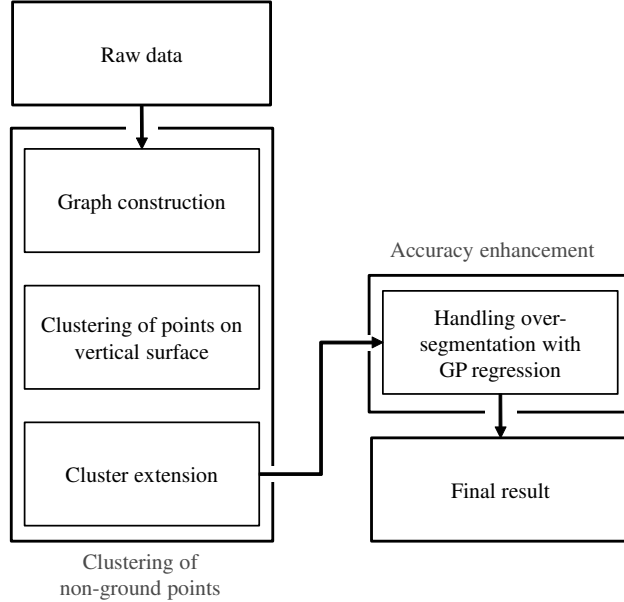


Figure 2.2 Processing flow of the proposed segmentation algorithm.

able accuracy enhancement in real-time, our algorithm selects each pair of objects suspected to suffer from over-segmentation by considering their sizes, the numbers of their element points, and their adjacency to neighboring objects. Then a GP regression model estimates the surface information of one object, i.e. the function values of test points, by utilizing the nearest  $k$  points of the other neighboring object as training points. When the estimated function values are close to the actual values, the two objects are merged.

## 2.4 Clustering of Non-ground Points

In this section, we describe the clustering process for non-ground points based on a combination of a 2D grid and an undirected graph structure. As aforemen-

tioned, our algorithm separates non-ground objects without ground extraction. Most ground points are not involved in the computation of Euclidean-distance, which significantly reduces unnecessary computation. As a result, our algorithm achieves higher stability on bumpy roads in urban driving environments as well as a real-time processing rate. Also, pointwise clustering based on the graph structure is implemented, where the dimensionality reduction that causes under-segmentation is not required.

#### 2.4.1 Graph Construction

To deal with the large amount of unordered points provided by the 3D LIDAR (up to 100,000 points per scanning), we construct an undirected graph,  $G = \{V, E\}$ , which enables fast access to the adjoining points for clustering in the following subsections. According to the method of [31], each measurement point is defined as a node, and the node is connected to four neighboring nodes by edges. Horizontal neighbors are both the left and right consecutive measurements of the same layer that denotes a series of points measured by the same laser. In addition, vertical neighbors are those in the adjacent upper and lower layers, where the yaw angles of the neighbors are almost equal.

#### 2.4.2 Clustering of Points on Vertical Surface

The next task described in this subsection is to cluster non-ground points without ground extraction. This task is inspired by an observation that most non-ground objects that should be recognized for safe driving are distinctly higher than the ground. Given a point cloud,  $P = \{p_i | p_i = (x, y, z)_i, i = 0, 1, \dots, N_P - 1\}$ , we employ

an evenly divided 2D grid plane to detect points of high elevation on varying slopes of an urban driving environment. The  $N_P$  points are assigned to proper cells of the grid according to their  $x - y$  coordinate, where the ground height of the  $j^{th}$  cell,  $h_j$ , is found by taking the lowest elevation value  $z$  among the assigned points. Then, the points above the corresponding ground heights are defined as a set of candidate points of object,  $C$ , which can be represented as:

$$C = \{p_i | z_i > h_j + \tau_h, p_i \in P\}, \quad (2.1)$$

where  $j$  indicates the cell index that covers the  $x - y$  coordinate of the  $p_i$  and  $\tau_h$  is a threshold value determining the minimum height for a non-ground point. Our grid plane is similar to that in other works [5, 18–21], whereby every point is assigned to a specific cell covering its  $x - y$  coordinate; however, dimension reduction is not implemented in this chapter.

Points in  $C$  are clustered with the neighboring points of the same layer as shown in Algorithm 1, where  $N_{PH}$  denotes the number of points per layer. The algorithm produces a set of line segments  $\mathcal{L}$  as follows:

$$\mathcal{L} = \{L_{s(v)} | v = 0, 1, \dots, N_{PV} - 1\}, \quad (2.2)$$

---

**Algorithm 1:** Line Segment Generation

---

**Input** : A set of candidate points of objects,  $C$

**Output:** A set of line segments per layer,  $\mathcal{L}$

```
for  $v = 0$  to  $N_{PV} - 1$  do
     $L_{s(v)} \leftarrow \emptyset$ ;
     $clusterflag = 0$ ;
     $k = 0$ ;
    for  $u = 0$  to  $N_{PH}$  do
        if  $p_{u,v} \in C$  and  $p_{u+1,v} \in C$  and  $a_{u,v} > \tau_s$  and  $a_{u+1,v} > \tau_s$  and
             $d(p_{u,v}, p_{u+1,v}) < \tau_d$  then
            if  $clusterflag = 0$  then
                 $startidx = u$ ;
                 $clusterflag = 1$ ;
            end
        else
            if  $clusterflag = 1$  and  $u - startidx + 1 > minpts$  then
                 $endidx = u$ ;
                 $clusterflag = 0$ ;
                 $s_{v(k)} = (startidx, endidx)$ ;
                 $L_{s(v)} = L_{s(v)} \cup \{s_{v(k)}\}$ ;
                 $k = k + 1$ ;
            end
        end
    end
end
```

---

$$L_{s(v)} = \{s_{v(k)} | k = 0, 1, \dots, N_{s(v)} - 1\}, \quad (2.3)$$

where  $N_{PV}$  is the number of the total layer of the 3D LIDAR,  $L_{s(v)}$  is the set of line segments in the  $v^{th}$  layer, and  $N_{s(v)}$  denotes the cardinality of  $L_{s(v)}$ . Let  $p_{u,v}$  be the  $u^{th}$  point in the  $v^{th}$  layer and  $N_{PH}$  be the same in all layers, where  $p_i = p_{u+v \cdot N_{PH}}$ . In each layer, clustering is performed when a current point  $p_{u,v}$  and a target point  $p_{u+1,v}$  satisfy the five conditions in Line 6 of Algorithm 1. First,



both points should be elements of  $C$ . Second, their slopes  $a_{u,v}$  and  $a_{u+1,v}$  should be steep enough to be more than a threshold  $\tau_s$ , where the slope  $a_{u,v}$  of  $p_{u,v}$  is calculated as follows:

$$a_{u,v} = \left| \frac{\Delta z}{\Delta r} \right|, \quad (2.4)$$

where  $\Delta z$  and  $\Delta r$  are the differences of  $z$  and  $r = \sqrt{x^2 + y^2}$  between  $p_{u,v}$  and  $p_{u,v-1}$ , respectively. Third,  $d(p_{u,v}, p_{u+1,v})$  should be closer than a threshold  $\tau_d$ , where  $d(p_{u,v}, p_{u+1,v})$  is a Euclidean-distance between the two points. The purpose of the slope condition is to filter out false positives that cannot be handled with the height threshold  $\tau_h$  alone, such as points on a steep road. The condition also retains the points on a flat surface of a non-ground object from being clustered, such as a roof of a vehicle, which will be dealt with in the next subsection. The  $k^{th}$  line segment in the  $v^{th}$  layer, i.e.  $s_{v(k)}$ , is stored in the form of  $(startidx, endidx)$  as shown in Line 15 of Algorithm 1. Line segments that have few element points that are less than  $minpts$  are removed to reduce false positives. Fig. 2.3(a) illustrates an example of the clustering process in the  $v^{th}$  layer. In the segmentation method of [33], Jump Distance Clustering (JDC) that at first sight seems similar to our Algorithm 1 is proposed. However, JDC only considers the distance between points, which is distinct from our Algorithm 1, where the slope condition is also considered.

Neighboring line segments are then merged in a vertical direction to produce chunks of points representing non-ground objects. The line segments in the adjacent layer are grouped together if the distance between their end points is shorter than  $\tau_d$  as in Fig. 2.3(b), where points on the topmost layer are not yet grouped because

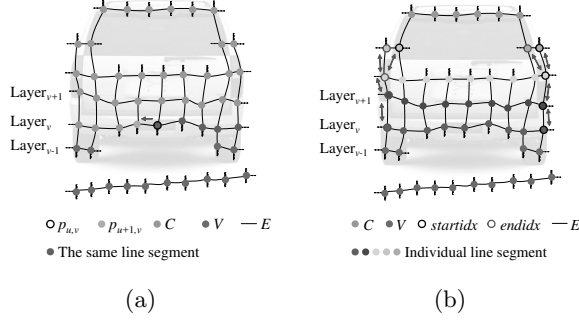


Figure 2.3 Clustering of points on vertical surface. (a) In each layer, line segments are generated by clustering  $p_{u,v} \in C$  toward horizontal direction. (b) Neighboring line segments are merged if a distance between their end points is shorter than  $\tau_d$ .

their slopes are less than  $\tau_s$ . Finally, a set of objects is generated as follows:

$$\mathbf{O} = \{O_t | t = 0, 1, \dots, N_S - 1\}, \quad (2.5)$$

where  $O_t$  is a segmented object and  $N_S$  is the number of objects. Among the objects in close range and low height, the long and thin objects are removed because they are considered as false positives on curbs.

### 2.4.3 Cluster Extension

With the segmentation result of the previous subsection, the precise object information of the 3D shape and size cannot yet be captured because only those points on a vertical surface that are higher than the ground by  $\tau_h$  are clustered. The goal in the next task described in this subsection is twofold. First, we extend the lower boundaries of all objects determined by  $h_j + \tau_h$  downward until they meet the ground. Then, we add points on the flat surface, such as the roof of a vehicle, to the adjacent point groups.

---

**Algorithm 2:** Extension of Lower Boundaries

---

**Input** : A set of segmented objects,  $\mathbf{O} = \{O_0, O_1, \dots, O_{N_S-1}\}$

**Output:** A set of extended objects,  $\mathcal{O} = \{O_{e(0)}, O_{e(1)}, \dots, O_{e(N_S-1)}\}$

```
for  $t = 0$  to  $N_S - 1$  do
     $C_{b(t)} \leftarrow \{p_{u,v} | p_{u,v} \in O_t \text{ and } p_{u,v-1} \notin C\};$ 
     $C_{bn(t)} \leftarrow \emptyset;$ 
     $O_{e(t)} \leftarrow \emptyset;$ 
    while  $C_{b(t)} \neq \emptyset$  do
         $termin\_cnt = 0;$ 
        forall  $p_{u,v} \in C_{b(t)}$  do
            if  $v - 1 > 0$  and  $p_{u,v-1} \notin C_{bn(t)}$  and  $a_{u,v-1} > \tau_s$  and
                 $d(p_{u,v}, p_{u,v-1}) < \tau_d$  then
                     $C_{bn(t)} \cup \{p_{u,v-1}\};$ 
                     $startidx = u;$ 
                    while  $p_{u+1,v-1} \notin C_{bn(t)}$  and  $a_{u+1,v-1} > \tau_s$  and
                         $d(p_{u,v-1}, p_{u+1,v-1}) < \tau_d$  do
                             $C_{bn(t)} \cup \{p_{u+1,v-1}\};$ 
                             $u = u + 1;$ 
                        end
                     $endidx = u;$ 
                     $s_{v-1(N_{s(v-1)})} = (startidx, endidx);$ 
                     $L_{s(v-1)} = L_{s(v-1)} \cup s_{v-1(N_{s(v-1)})};$ 
                     $N_{s(v-1)} = N_{s(v-1)} + 1;$ 
                else if  $a_{u+1,v-1} \leq \tau_s$  then
                     $termin\_cnt = termin\_cnt + 1;$ 
                end
            end
        end
         $O_t \cup C_{bn(t)};$ 
         $C_{b(t)} \leftarrow C_{bn(t)};$ 
         $C_{bn(t)} \leftarrow \emptyset;$ 
        if  $termin\_cnt / |C_{b(t)}| > \tau_t$  then
            break;
        end
    end
     $O_{e(t)} \leftarrow O_t;$ 
end
```

---

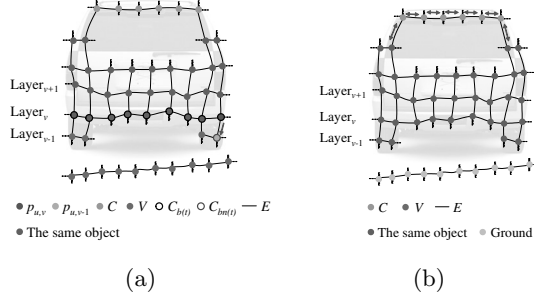


Figure 2.4 Cluster extension. (a) Lower boundary between an object and ground is detected by confirming slopes of vertical neighbor nodes. (b) After the lower boundaries are determined, remaining  $p_{u,v} \in C$  are merged into the nearest objects based on Euclidean-distance.

The lower boundaries of all objects detected in the previous subsection are extended as in Algorithm 2. For each object  $O_t$ , sets of points on the lower boundary  $C_{b(t)}$  and the extended boundary  $C_{bn(t)}$  are defined as follows:

$$C_{b(t)} = \{p_{u,v} | p_{u,v} \in O_t \text{ and } p_{u,v-1} \notin C\}, \quad (2.6)$$

$$C_{bn(t)} = \emptyset. \quad (2.7)$$

Then,  $a_{u,v-1}$  and  $d(p_{u,v}, p_{u,v-1})$  are computed for a point  $p_{u,v} \in C_{b(t)}$ . When  $a_{u,v-1} > \tau_s$  and  $d(p_{u,v}, p_{u,v-1}) < \tau_d$ , the point  $p_{u,v-1}$  is added to  $C_{bn(t)}$  as depicted in Fig. 2.4(a), which is followed by clustering toward the horizontal direction as indicated in the lines between Line 9 and Line 18 of Algorithm 2. After all points in  $C_{b(t)}$  are analyzed,  $p_{u,v} \in C_{bn(t)}$  is added to  $O_t$ ,  $C_{b(t)}$  is substituted for  $C_{bn(t)}$ , and a new iteration starts, i.e. from Line 5 to Line 29. In the middle of the procedure, Line 26 terminates to extend a lower boundary of  $O_t$  when the portion of  $p_{u,v} \in C_{b(t)}$  that violates the slope condition in Line 8 exceeds  $\tau_t$ . When Algorithm 2 is applied

to all of  $O_t \in \mathbf{O}$ , a set of extended objects  $\mathcal{O}$  is produced as follows:

$$\mathcal{O} = \{O_{e(t)} | t = 0, 1, \dots, N_S - 1\}. \quad (2.8)$$

The remaining  $p_{u,v} \notin C$  are all considered as ground points.

Some points of  $p_{u,v} \in C$  are still not assigned to any of the  $O_{e(t)}$ . They could be points on a flat surface of a non-ground object that should be added to the nearest  $O_{e(t)}$ , as the points on the topmost layer in Fig. 2.4(b), or a small fraction of a steep road that should be filtered out. These points are added to the nearest chunk of points, where the points adjacent to both a non-ground object and the ground are added to the ground.

## 2.5 Accuracy Enhancement

The segmentation result from chapter 2.4 might include some over-segmented objects as in Fig. 2.5(a). The over-segmented portions decrease the precision of the segmentation algorithm because they increase the number of detected objects, the same as false positives. Moreover, the separated portions degrade the performance of another perception step, such as tracking, that utilizes the result of the segmentation as input data by generating false tracks or disturbing the data association procedure.

Over-segmentation arises from irregular large gaps between measurement points. The reasons are twofold. First, the different slopes of an object surface, such as a roof, a bumper, or a window of a vehicle, change the gaps. Second, the lasers of 3D LIDARs are often not returned to the receiving device on some specific types of surfaces, e.g. surfaces made of glass or painted in a black color, which enlarges

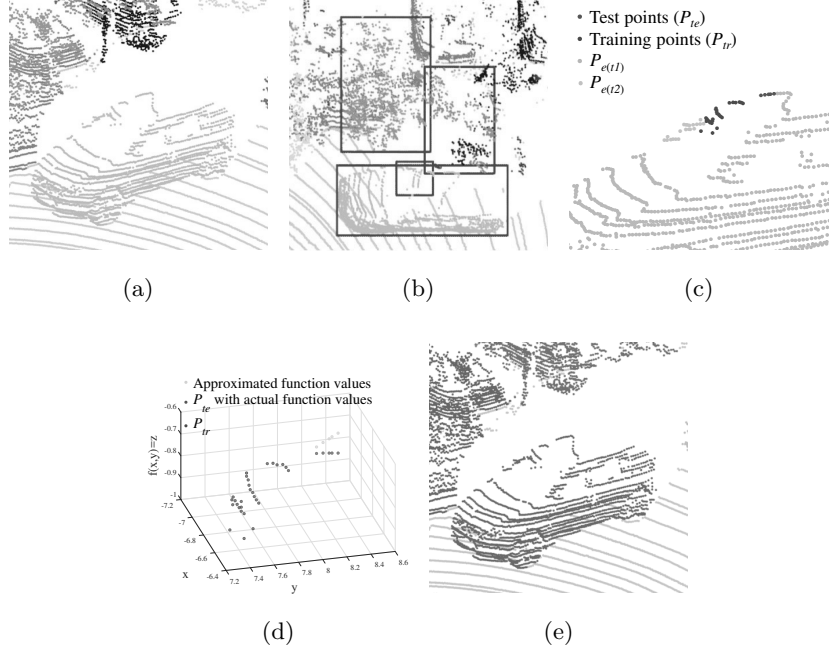


Figure 2.5 Entire process of over-segmentation handling. (a) Example of over-segmentation, shown in orange and light green. (b) Overlapping areas of neighboring objects in the  $x - y$  plane. (c) Test points (red) and training points (blue) of the paired objects  $P_{e(t_1)}$  (orange) and  $P_{e(t_2)}$  (light green). (d) Example of GP regression in a Cartesian coordinate.  $z$  coordinates of the light blue points are  $\bar{\mathbf{f}}_*$  approximated by Eq. (2.19) and  $z$  coordinates of the red points are  $\mathbf{f}_{actual}$ . (e) Result of over-segmentation handling. Only portions of the vehicle are merged, although a couple of adjacent objects were overlapped in (b).

the gaps. Consequently, points of the same object are not clustered together when using methods based on a constant Euclidean-distance, as employed in numerous researches [1, 2, 16–21, 33]. In this section, we describe our approach in which a regression problem is designed to handle the over-segmentation problem.

### 2.5.1 Approach to Handling Over-segmentation

In a driving environment, most over-segmentations arise at vehicles because they typically meet at least one of the two conditions for generating the aforementioned over-segmentation. Particularly, a vehicle tends to be separated into upper and lower parts across the window, where the separated portions share an area overlapped in the  $x - y$  plane of a Cartesian coordinate. We therefore first find a pair of objects that overlap in the  $x - y$  plane, and then merge the two objects if some conditions are satisfied. To determine whether the two objects should be merged, we design a regression problem, which estimates the surface coordinate values of one object as if they are unknown by regarding those of the other object as observation. When the estimated values are close to the actual values, we combine the two objects with the assumption that consistency of the same object surface is proven by the regression. Otherwise, the two objects are not merged, based on the decision that they lack consistency of the same object surface because their surface coordinate values cannot be approximated based on those of each other. Fig. 2.5(e) illustrates the result whereby two portions of a vehicle are merged; however, a tree colored violet is not merged with the vehicle, even though the tree and the vehicle share an area overlapped in the  $x - y$  plane as shown in Fig. 2.5(b).

### 2.5.2 Handling Over-segmentation with GP Regression

To obtain the best result using the approach outlined in the previous subsection, performing high accuracy regression is of great importance. In this chapter, we employ GP regression due to its promising ability to model data correlation. A GP is a generalization of the Gaussian probability distribution over random variables, where each random variable is considered as the function value  $f(\mathbf{x})$  at a specific input  $\mathbf{x}$  [12]. A GP is specified by a mean function  $m(\mathbf{x})$  and a covariance function  $k(\mathbf{x}, \mathbf{x}')$  as follows:

$$f(\mathbf{x}) \sim GP(m(\mathbf{x}), k(\mathbf{x}, \mathbf{x}')), \quad (2.9)$$

where the mean function is usually taken to be zero for notational simplicity, and the covariance function, also called a kernel, models the correlation between the function values. Without any assumption of the function, e.g. linear or quadratic, GP regression approximates unknown function values well by taking into account the correlation of the function values and the given observation, even in abrupt changes. This feature facilitates the flexible handling of the over-segmentation that occurs in a very irregular form in a driving environment.

In a GP framework, the 3D points dealt with in this chapter form two types of an input  $\mathbf{x}$  and the corresponding function value  $f(\mathbf{x})$  as follows:

$$f(\mathbf{x}) = f(x, y) = z, \quad (2.10)$$

$$f(\mathbf{x}) = f(\theta, \phi) = r, \quad (2.11)$$

where Eq. (2.10) and Eq. (2.11) are the representations of the 3D points in a Cartesian coordinate system and a spherical coordinate system, respectively. In this



subsection, we show how the correlation between the points of the over-segmented objects is modeled and the correlation is utilized to merge the over-segmented objects based on GP regression.

Applying GP regression to all objects of the segmentation result  $\mathcal{O}$  is time-consuming and prone to increase under-segmentation in the case of incorrect operation. Since our concern is to complement the over-segmentations of vehicles, extremely large or small objects are excluded. The number of line segments of each object is also considered because it is useful to filter out bushes that have significantly more line segments than vehicles. Lastly, we draw a 2D bounding box of each segmented object as shown in Fig. 2.5(b), where two objects overlapping each other in the  $x - y$  plane are paired to be analyzed by GP regression. Given a pair of objects  $O_{e(t_1)}$  and  $O_{e(t_2)}$ , the point sets of the two objects are defined as follows:

$$P_{e(t_1)} = \{p_i | p_i \in O_{e(t_1)}\}, \quad (2.12)$$

$$P_{e(t_2)} = \{p_i | p_i \in O_{e(t_2)}\}. \quad (2.13)$$

Our algorithm selects  $n$  training points from one object as observation and  $n_*$  test points from the other object of which the function values will be estimated based on the observation. To minimize the distances between the test points and the training points, a pair of the nearest points between the two objects are first determined as follows:

$$(p_{te}, p_{tr}) = \arg \min_{p_{te} \in P_{e(t_1)}, p_{tr} \in P_{e(t_2)}} d(p_{te}, p_{tr}), \quad (2.14)$$

where  $p_{te}$  and  $p_{tr}$  become the first elements of a test set  $P_{te}$  and a training set  $P_{tr}$ , respectively. Then, the  $P_{te}$  and  $P_{tr}$  elements are augmented by containing the

nearest  $n_* - 1$  points of  $p_{te}$  in  $P_{e(t_1)}$  and the nearest  $n - 1$  points of  $p_{tr}$  in  $P_{e(t_2)}$ , respectively. Fig. 2.5(c) illustrates  $P_{te}$  and  $P_{tr}$  of the paired objects  $P_{e(t_1)}$  and  $P_{e(t_2)}$ .

With the training set of  $n$  observations  $P_{tr} = \{(\mathbf{x}_i, f(\mathbf{x}_i)) | i = 0, \dots, n - 1\}$  and the test set of  $n_*$  points  $P_{te} = \{(\mathbf{x}_{j(*)}, f_*(\mathbf{x}_{j(*)})) | j = 0, \dots, n_* - 1\}$ , of which function values  $\mathbf{f}_*$  are assumed to be unknown, performing GP regression first requires a joint distribution of  $\mathbf{f}$  and  $\mathbf{f}_*$ . The joint distribution incorporates the correlation of the training points and the test points, where the correlation is modeled by a squared exponential covariance function that has the following form in this chapter:

$$k(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \exp \left[ -\frac{1}{2}(\mathbf{x} - \mathbf{x}')^T M (\mathbf{x} - \mathbf{x}') \right]. \quad (2.15)$$

$\sigma_f^2$  is a signal variance, and  $M = l^{-2}\mathbf{I}$  is a symmetric matrix, where  $l$  is a *characteristic length-scale* indicating the distance required for the function values to be uncorrelated in the directions along the axes of the input domains  $x_1$  and  $x_2$ . Sensor noise in the observations is assumed to be an additive independent identically distributed Gaussian with variance  $\sigma_n^2$ , which redefines the covariance function in the observation values as:

$$\text{cov}(f(\mathbf{x}_p), f(\mathbf{x}_q)) = k(\mathbf{x}_p, \mathbf{x}_q) + \sigma_n^2 \delta_{pq}, \quad (2.16)$$

where  $\delta_{pq}$  is a Kronecker delta that becomes 1 when  $p = q$  and 0 otherwise. The elements of  $\boldsymbol{\theta}_h = \{l, \sigma_f^2, \sigma_n^2\}$  are called hyperparameters that should be determined in a learning process. Then, the joint distribution of  $\mathbf{f}$  and  $\mathbf{f}_*$  is given by:

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{f}_* \end{bmatrix} \sim N \left( 0, \begin{bmatrix} K(X, X) + \sigma_n^2 I & K(X, X_*) \\ K(X_*, X) & K(X_*, X_*) \end{bmatrix} \right), \quad (2.17)$$

where  $X$  is a matrix of the  $n$  training points of which each column is each input vector and  $X_*$  is a matrix of the  $n_*$  test points. With the  $n$  training points and the

$n_*$  test points,  $K(X, X_*)$  is the  $n \times n_*$  matrix of the covariances calculated at all pairs of the training points and the test points. Other entries,  $K(X, X)$ ,  $K(X_*, X_*)$ , and  $K(X_*, X)$ , are computed in the same way.

Our goal is to approximate  $\mathbf{f}_*$  in  $P_{te}$  based on the training points in  $P_{tr}$ . This is represented as the following conditional distribution:

$$\mathbf{f}_*|X, \mathbf{f}, X_* \sim N(\bar{\mathbf{f}}_*, \text{cov}(\mathbf{f}_*)) , \quad (2.18)$$

where the mean is the best estimates for  $\mathbf{f}_*$ :

$$\bar{\mathbf{f}}_* = K(X_*, X) [K(X, X) + \sigma_n^2 I]^{-1} \mathbf{f}, \quad (2.19)$$

and the covariance gives the corresponding uncertainty:

$$\begin{aligned} \text{cov}(\mathbf{f}_*) = \\ K(X_*, X_*) - K(X_*, X) [K(X, X) + \sigma_n^2 I]^{-1} K(X, X_*). \end{aligned} \quad (2.20)$$

Fig. 2.5(d) shows the approximated values  $\bar{\mathbf{f}}_*$  by Eq. (2.19) and the actual function values of the test points  $\mathbf{f}_{actual}$  in a Cartesian coordinate. The root-mean-square error (RMSE) between  $\bar{\mathbf{f}}_*$  and  $\mathbf{f}_{actual}$  is represented as follows:

$$\text{var}(f_*) < \tau_v^2, \quad (2.21)$$

$$e_{RMSE} = \sqrt{\sum_{r=1}^{n_{**}} (\bar{f}_{*(r)} - f_{actual(r)})^2 / n_{**}}, \quad (2.22)$$

where  $\text{var}(f_*)$  is a variance of an estimated value  $f_*$  based on Eq. (2.20) and  $n_{**}$  is the number of the points with low uncertainty filtered by the threshold value  $\tau_v$  that has two values according to the coordinate types of the function values. The function values of large vehicles, such as buses or trucks, cannot be estimated well with Eq. (2.10) because the training points and the test points have similar input

vectors in the  $x - y$  plane. This leads to the introduction of Eq. (2.11) and  $\tau_{vp}$  of a degree unit for the large vehicles. For the other types of vehicles, such as sedans, small trucks, and SUVs, Eq. (2.10) and  $\tau_{vc}$  of a meter unit are utilized. To adopt Eq. (2.10) or Eq. (2.11) as appropriate from our segmentation result, where object labels are unknown, we draw their integrated bounding box, i.e., the bounding box of an object that will be generated if the paired objects are merged, before running GP regression. When any side of the bounding box in the  $x - y$  plane exceeds the typical length of a sedan, 6.0 m in this chapter, Eq. (2.11) is employed to perform GP regression regardless of whether the paired objects are actually large vehicles or not. Otherwise, Eq. (2.10) is utilized.

When  $e_{RMSE}$  is less than a predefined threshold  $\tau_r$ , the two paired objects are actually merged. If not, both test points and training points are selected in the opposite object, and GP regression is performed again. Fig. 2.5(e) is the final result of over-segmentation handling. Although there are more than two overlapping areas due to the tree canopies as shown in Fig. 2.5(b), the two certain portions of the vehicle are merged.

### 2.5.3 Learning Hyperparameters

Finding the optimal hyperparameters of the covariance function is a significant task in order to achieve a promising estimation performance of GP regression. Learning the hyperparameters  $\theta_h$  with the given training data set is carried out by maximizing the following log marginal likelihood [12]:

$$\log p(\mathbf{y}|X, \theta_h) = -\frac{1}{2}\mathbf{y}^T K_y^{-1} \mathbf{y} - \frac{1}{2} \log |K_y| - \frac{n}{2} \log 2\pi, \quad (2.23)$$

where  $\mathbf{y}$  is the output, or the function value, of the training data at an input  $\mathbf{x}$ ,  $K_y = K_f + \sigma_n^2 I$  is the covariance matrix for noisy data, and  $K_f$  is the covariance matrix of noise-free data computed by Eq. (2.15).

For hyperparameter learning, we prepared special sets of point clouds for the learning in this study. Since most over-segmented objects were in the 20 - 40 meters (m) range in our experiments, point clouds of an SUV and a bus segmented successfully in the range were chosen as the special sets. Although the two sets for the learning cannot represent all over-segmentation cases, it turns out that the hyperparameters learned with them work well, as will be demonstrated in the next section.

## 2.6 Experiments

We carried out intensive experiments to verify the contributions of our algorithm. Stability, capability of real-time processing, and improvement of segmentation and tracking accuracy by over-segmentation handling are discussed in this section. To compare the performance of the proposed algorithm running without ground extraction in various driving environments, two segmentation algorithms implementing ground extraction were tested together: a mesh-based algorithm from the most cited study [1] and a polar grid-based algorithm showing the fastest processing rate [2] among the previous works mentioned in chapter 3.2.

### 2.6.1 Experiment Environment

We collected our own dataset from a campus road at Seoul National University. The total length of the track was around 5.2 kilometers, where the maximum

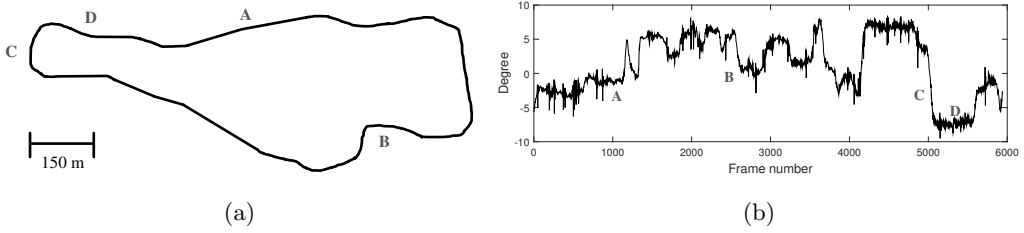


Figure 2.6 Experiment environment. (a) Campus road of SNU. (b) Pitch angle variation with the frame number.

difference of altitude was around 100 meters as shown in Fig. 2.6(a). The track included numerous irregular slopes and speed bumps, rendering it an appropriate environment with which to verify the reliability of our algorithm on various slope conditions and to determine the effects of faulty ground extractions on segmentation results in the other algorithms. For more various experiments, we also prepared two KITTI datasets [39].

Our experimental driving platform was equipped with a Velodyne HDL-64E S2 as a 3D LIDAR and an OXTS RT3002 as a high precision positioning system. Using the inertial measurements of the positioning system, we corrected the 3D point clouds of the LIDAR to reduce the influence of the vehicle’s movement while scanning, as in [18]. For visualization, we utilized Point Cloud Library (PCL) [40].

### 2.6.2 Evaluation Metrics

Evaluations of previous works [1, 22, 23] were implemented in a point unit, which required manual labeling of all points over 100,000 per frame. However, labeling all points is unsuitable for evaluating an algorithm for a number of frames, because it is excessively time-consuming. For this reason, in this chapter we propose new



Figure 2.7 Example of the *target objects*, colored green, in a single frame.

evaluation metrics of a segmentation algorithm for 3D point clouds, where the labeling is carried out in an object unit. In contrast to the previous works that utilized less than 10 frames for evaluation, we were able to quantify the performance of our algorithm for 1,600 frames with the aid of the new metrics.

We defined *target objects* that should be clearly separated from the other objects in each frame. All types of objects within the 70 meters range, e.g. vehicles, pedestrians, trees, bicycles, traffic signs, and traffic lights, with more than 30 points were considered as *target objects*. Large objects such as buildings and bushes, and occluded objects were excluded from the *target objects* because the criteria of successful segmentation were ambiguous. Fig. 2.7 illustrates an example of the *target objects* colored in green in a single frame data. The *target objects* were manually extracted in various slope conditions to compare the reliability of the competing algorithms, where the slope conditions were discovered by the pitch angles provided by our positioning system. Fig. 2.6(b) shows the variation of the pitch angles along with the frame number with marks indicating the regions utilized to produce the *target objects*. We prepared four scenarios that consisted of two flat roads and two

varying slopes, where the total number of *target objects* amounted to 21,803.

In addition to the over-segmentation and under-segmentation previously explained, we define three further states of the *target objects* in a segmentation result for evaluation: True Positives (TPs), False Positives (FPs), and False Negatives (FNs). The meaning of each term in this chapter is as follows:

- TPs are *target objects* successfully separated from the other objects. *Target objects* are considered as FNs or over-/under-segmentations when the element points are clustered incompletely.
- FNs are *target objects* that have been missed. *Target objects* are also considered as FNs when the element points are clustered by less than two thirds due to the ground extraction error in the competing algorithms or incorrect generation of  $C$  in our algorithm.
- FPs are objects generated from the ground. When a boundary of a *target object* violates the ground and the area is over one third of the *target object* in the  $x - y$  plane, it is also counted as an FP.

Five metrics are then defined to quantify each performance related to the segmentation state as follows:

$$\begin{aligned}\text{OSR} &= \frac{\text{TPs}}{\text{TPs} + \text{over-segmentation}}, \\ \text{Precision} &= \frac{\text{TPs}}{\text{TPs} + \text{FPs}}, \\ \text{e-Precision} &= \frac{\text{TPs}}{\text{TPs} + \text{FPs} + \text{over-segmentation}}, \\ \text{USR} &= \frac{\text{TPs}}{\text{TPs} + \text{under-segmentation}},\end{aligned}$$



$$\text{Recall} = \frac{\text{TPs}}{\text{TPs} + \text{FNs}},$$

where OSR, USR, and e-precision refer to the *Over-segmentation Suppression Rate*, *Under-segmentation Suppression Rate*, and *effective precision*, respectively. e-Precision is introduced to quantify the effect of over-segmentation on precision, whereby separated portions of the *target objects* increase the number of segmented objects in the actual driving situation, the same as FPs.

### 2.6.3 Processing Time

A total of 5,938 consecutive frames of our own dataset, including the four slope conditions in Fig. 2.6, were utilized to test the processing speeds of the three algorithms. Because there was no explicit standard of real-time processing, we defined 100 ms as the minimum requirement, corresponding to the sensor measurement update cycle. Fig. 2.8 illustrates the processing times along with the frame number, while Table 2.2 presents the means, standard deviations (SD), and maximum and minimum times of the three results. The mesh-based algorithm [1] fails to achieve a real-time processing rate, with the longest mean time of 115.5 ms. The ground extraction of the mesh-based algorithm requires computations to find the gradients of all points, which severely lowers the overall processing time. The polar grid-based algorithm [2] utilized various techniques to extract the ground quickly; the generation of a 2D polar grid, dimensionality reduction, use of a prototype point of each cell, and CCL performed on the 2D grid. Consequently, a mean time of 41.8 ms was shown, similar to that of 40.1 ms in our work, where both algorithms achieved real-time performances. Note that our fast segmentation is mainly due to omitting the ground extraction that is realized by clustering non-ground points in

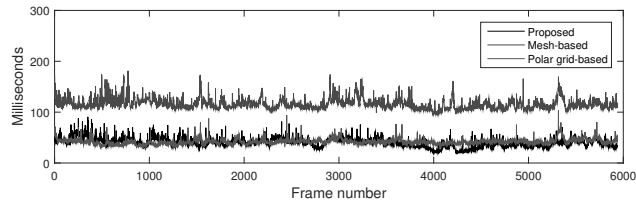


Figure 2.8 Comparison of the processing times for 5938 frames from our own data set on an Intel Core i7-4770K at 3.5 GHz.

Table 2.2 Processing time on our own dataset.

Algorithm	Ours	[1]	[2]
Mean (ms)	40.1	115.5	41.8
Standard deviation (ms)	8.4	10.6	5.2
Maximum (ms)	90.8	185.3	103.2
Minimum (ms)	20.7	94.4	31.3

$C$  only. Thus, as the number of points in  $C$  increases, the required processing time increases, which results in a higher SD than that for the polar grid-based algorithm by 3.2 ms. However, our maximum and minimum times are still less than those of the polar grid-based algorithm. In the next subsection, our greater segmentation accuracy will also be demonstrated.

#### 2.6.4 Accuracy on Various Driving Environments

Table 2.3 lists the values of the thresholds and the hyperparameters of GP in this chapter, where  $\theta_{hc}$  is a vector of the hyperparameters for small vehicles, and  $\theta_{hp}$  is the hyperparameters for large vehicles. Except for  $\theta_{hc}$  and  $\theta_{hp}$ , which are

Table 2.3 Parameter setting.

Parameter	Grid size	$\tau_h$	$\tau_s$	$\tau_t$	$\tau_d$	$\tau_{vc}$	$\tau_{vp}$	$\tau_r$	$\theta_{hc}$	$\theta_{hp}$
Value	$2.0 \times 2.0$	0.3	0.4	0.2	0.5	1.5	12	0.5	{0.8202, 1.7104, 0.1734}	{1.1907, 6.4747, 0.0464}

Table 2.4 Accuracy comparison on our own dataset.

Road type	A (Flat road)			B (Slope)			C (Slope)			D (Flat road)			Total		
Frames	200			200			200			200			800		
# of Targets	3963			2420			1680			3187			11250		
Algorithm	Ours	[1]	[2]	Ours	[1]	[2]	Ours	[1]	[2]	Ours	[1]	[2]	Ours	[1]	[2]
Over-seg.	19	117	181	57	110	145	62	107	107	44	127	147	182	461	580
FPS	58	410	196	87	62	154	100	95	112	69	353	123	314	920	585
Under-seg.	6	18	32	216	189	245	161	127	146	68	57	56	451	391	479
FNS	207	181	156	105	209	234	51	47	15	74	99	99	437	536	504
TPs	3676	3600	3600	1979	1925	1793	1360	1437	1369	2969	2906	2846	9984	9868	9608
OSR	<b>0.995</b>	0.969	0.952	<b>0.972</b>	0.946	0.925	<b>0.956</b>	0.931	0.928	<b>0.985</b>	0.958	0.951	<b>0.982</b>	0.955	0.943
Precision	<b>0.984</b>	0.898	0.948	0.958	<b>0.969</b>	0.921	0.932	<b>0.938</b>	0.924	<b>0.977</b>	0.892	0.959	<b>0.970</b>	0.915	0.943
e-Precision	<b>0.979</b>	0.872	0.905	<b>0.932</b>	0.918	0.857	<b>0.894</b>	0.877	0.862	<b>0.963</b>	0.858	0.913	<b>0.953</b>	0.877	0.892
USR	<b>0.998</b>	0.995	0.991	0.902	<b>0.911</b>	0.880	0.894	<b>0.919</b>	0.904	0.978	<b>0.981</b>	<b>0.981</b>	0.957	<b>0.962</b>	0.953
Recall	0.947	0.952	<b>0.958</b>	<b>0.950</b>	0.902	0.885	0.964	0.968	<b>0.989</b>	<b>0.976</b>	0.967	0.966	<b>0.958</b>	0.948	0.950

determined by maximization of Eq. (2.23), we manually set those values considering their function.  $\tau_d$ ,  $\tau_{vc}$ ,  $\tau_{vp}$ , and  $\tau_r$  are related to the gaps between the non-ground points on the same object. Thus, they are set not to merge distant points. Grid size and  $\tau_h$  determine the generation of  $C$  and should be adjusted according to the size of objects of interest. Grid size  $2.0 \times 2.0 \text{ m}^2$  was chosen by considering that vehicle widths were typically less than 2.0 m.  $\tau_h$  was set to 0.3 because the minimum height of objects in our experiment data was approximately 0.4 m.  $\tau_s$  and  $\tau_t$  determine the lower boundaries of objects. Since the slope of a boundary point is lowered due to the enlarged gap between layers for a distant range,  $\tau_s$  is small. Similarly,  $\tau_t$  is

also small because the number of points on the object boundary is lowered in the distant range.

Table 2.4 presents an accuracy comparison of the three algorithms from our own dataset. Our algorithm outperforms the other two algorithms in terms of over-segmentation handling on every slope condition, which results in a higher OSR than algorithms [1] and [2] by 2.7% and 3.9% on average, respectively. Our algorithm also shows higher rates in most other metrics, except for a USR that is slightly lower than that of the mesh-based algorithm [1]. This can be seen as the result of faulty over-segmentation handling, however, another type of error also increased the number of under-segmentations, as will be clarified below. The number of *target objects* in the table is less than the sum of the five states of the segmentation results. The reasons for this are twofold. First, most FPs were not related to the *target objects* because they were generated from the ground where there were no *target objects*. Second, objects suffering from both over- and under-segmentation at the same time increased the number of over- and under-segmentation together. Fig. 2.9 shows comparisons of over-segmentation handling in the three algorithms and Fig. 2.10 illustrates the segmentation results of our algorithm, showing the presence of various *target objects*.

To verify the reliability of our algorithm which does not require ground extraction in various driving environments, we compared the performances on flat roads and slopes in Table 2.5. The numerical values from the second to the fourth columns represent the sum of the results on the flat roads A and D in Table 2.4, and those from the fifth to the seventh columns represent the sum of the results on the slopes B and C. On flat roads, the three algorithms all show better re-

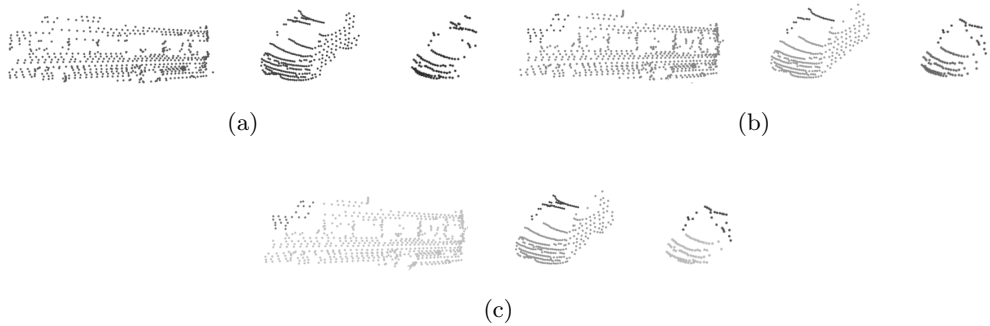


Figure 2.9 Comparisons of over-segmentation handling results. A bus, an SUV, and a sedan are represented in each row from the left. (a) Our algorithm. (b) Mesh-based algorithm [1]. (c) Polar grid-based algorithm [2].

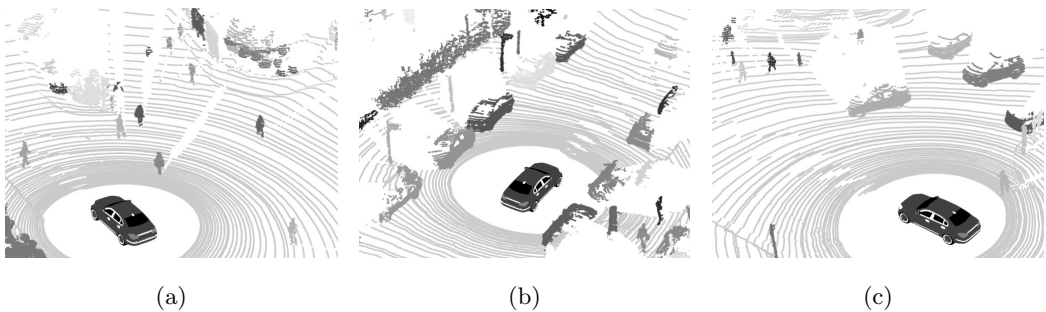


Figure 2.10 Segmentation results of the proposed algorithm. Most *target objects* are (a) pedestrians, (b) vehicles, and (c) pedestrians and vehicles at an intersection. Each color represents an object separated, while dark yellow indicates the ground.

Table 2.5 Contextual performance variation on our own dataset.

Scenarios	Flat roads (A,D)			Slopes (B,C)		
Frames	400			400		
# of <i>Targets</i>	7150			4100		
Algorithm	Ours	[1]	[2]	Ours	[1]	[2]
Over-seg.	63	244	328	119	217	252
FPS	127	763	319	187	157	266
Under-seg.	74	75	88	377	316	391
FNs	281	280	255	156	256	249
TPs	6645	6506	6446	3339	3362	3162
OSR	<b>0.991</b>	0.964	0.952	<b>0.966</b>	0.939	0.926
Precision	<b>0.981</b>	0.895	0.953	0.947	<b>0.955</b>	0.922
e-Precision	<b>0.972</b>	0.866	0.909	<b>0.916</b>	0.900	0.859
USR	<b>0.989</b>	<b>0.989</b>	0.987	0.899	<b>0.914</b>	0.890
Recall	0.959	0.959	<b>0.962</b>	<b>0.955</b>	0.929	0.927

sults than those on slopes. Particularly, our algorithm achieves higher rates than the other algorithms in most metrics with the aid of over-segmentation handling, where only recall is slightly lower than the polar grid-based algorithm. The extent of performance variance of our work on slopes is also acceptable compared with the other algorithms. On slopes, every algorithm shows performance degradation. The mesh-based and the polar grid-based algorithms exposed worse ground extraction, and our algorithm was affected by the incorrect generation of  $C$ . The tendency resulting from the degradation depended on each algorithm. The FPS on curbs were increased in the proposed algorithm and the polar grid-based algorithm, which also augmented under-segmentations because the FPS connected different *target objects* on the road, such as street lights. Consequently, the precision and USRs of our

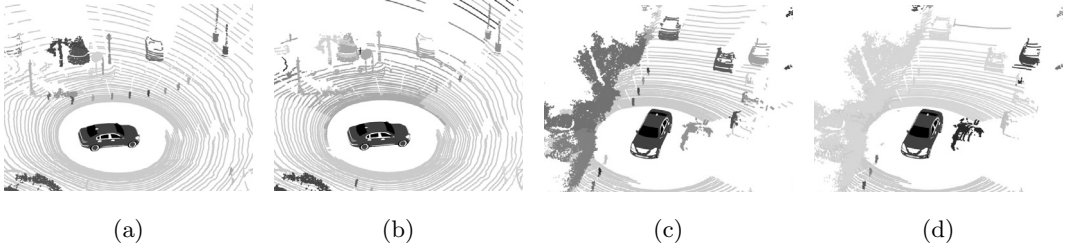


Figure 2.11 Illustration of ground extraction failure. (a)-(b) and (c)-(d) are images of segmentation results of the same frame, respectively. In contrast to our algorithm in (a) and (c), the pink and the light green areas in (b) and (d) are totally distorted due to the faulty ground extraction. (b) is the result of the polar grid-based algorithm [2] and (d) is the result of the mesh-based algorithm [1].

algorithm and the polar grid-based algorithm were reduced compared to the result of the mesh-based algorithm, whereas the mesh-based algorithm shows the lower recall than our algorithm due to the higher FNs. Considering the higher OSR and e-precision that functions as actual precision in driving conditions, our algorithm is still competitive.

We rather demonstrated that our algorithm has better stability, as presented in Fig. 2.11. In the worst case, the ground extractions implemented in the other two algorithms failed, which increased the *target objects* not segmented at all in some directions because clustering of non-ground points was performed based on the ground extraction result. The ground extraction failure occurs because the parameters optimized for ground extraction cannot fit in all driving conditions, especially on bumpy roads, which did not appear in our work. Some threshold values in our algorithm, such as 2D grid size,  $\tau_h$ , and  $\tau_s$ , were also affected by

the driving conditions. We carefully verified the robustness of all parameters in Table 2.3 including the three parameters in various driving conditions, such as uphill roads, downhill roads, bumpy roads, roads with curb, roads near grass, and intersections. Particularly, pitch angles in our dataset continuously varied over 18 degrees. Considering the maximum difference of the pitch angles in the KITTI dataset was less than 5 degrees (we randomly checked six datasets including two sets that will be discussed in this subsection), the pitch angle variation of our own dataset can be regarded as sufficiently large. Thus, all the parameters in our algorithm can be considered as robust to various driving conditions. In addition, the effects of the inappropriate threshold values in our algorithm are limited to individual objects only, which is a distinct feature compared with the previous studies where ground extraction error tends to spread. This feature provides our algorithm with more stable performance than the other two algorithms, regardless of the slope conditions.

For evaluation in other types of driving environments, we performed additional experiments using two KITTI datasets. In a city dataset, many road structures such as street lights and trees appeared on grass. In a road dataset, all *target objects* were only vehicles. Slope conditions were not considered since the maximum variation of pitch angles of the KITTI dataset was less than 5 degrees, as discussed above. Table 2.6 shows the segmentation result where only the polar grid-based algorithm was compared, because the mesh-based algorithm failed to meet the real-time constraint as discussed in the previous subsection. To demonstrate the robustness of the parameters, we set all parameters of the tested algorithms to be the same as those of our own dataset. In the total result, our algorithm outperformed the polar



Table 2.6 Accuracy comparison on the KITTI dataset.

Scenarios	City		Road		Total	
Frames	400		400		800	
# of <i>Targets</i>	6839		3714		10553	
Algorithm	Ours	[2]	Ours	[2]	Ours	[2]
Over-seg.	55	318	206	449	261	767
FPS	65	522	3	21	68	543
Under-seg.	6	34	4	8	10	42
FNs	222	118	132	41	354	159
TPs	6517	5939	3372	3212	9889	9151
OSR	0.992	0.949	0.942	0.877	0.974	0.923
Precision	0.99	0.919	0.999	0.994	0.993	0.944
e-Precision	0.982	0.876	0.942	0.872	0.968	0.875
USR	0.999	0.994	0.999	0.998	0.999	0.995
Recall	0.967	0.981	0.962	0.987	0.965	0.983

grid-based algorithm in all the metrics, except for recall, which was lowered due to the low point density of distant objects. Fundamentally, our algorithm generates separated objects by merging neighboring line segments. However, line segments are not generated well when only one point is returned from the surfaces of thin poles. We set the *minpts* in Algorithm 1 to 1 for speedy computation and reduction of FPS, which caused large gaps between the line segments of the distant thin poles, particularly in the city environment. A similar problem arose for pillars near vehicle windows in the road environment. For the distant range, the large gap between line segments separated vehicles into too small parts to be coped with our over-segmentation handling process. Consequently, they were incorrectly removed,

which increased FNs. In the case of the polar grid-based algorithm, almost eight times higher FPs were generated than those of our algorithm in total. The reason was the incorrect ground extraction on grass areas in the city environment, which did not occur in our algorithm. Also, the polar grid-based algorithm suffered from over-segmentation of distant poles due to empty voxels between points, which resulted in low OSR, although comparatively fewer vehicles appeared in the city data than in the road data.

We note that the focus of this chapter is segmentation in normal driving environments, such as cities, highways, and countryside. We expect the performance variation of the proposed algorithm in those environments will not be severe with the parameter settings in Table 2.3. The performance of the proposed algorithm can be degraded in environments of field or terrain where explicit roads are absent, potentially yielding a steep hill identified as an object or objects regarded as parts of the steep hill. Segmentation in that type of the extreme off road lies beyond the scope of our study.

The generation of  $C$  based on the 2D grid that enables our algorithm to fulfill clustering of non-ground points without ground extraction creates two types of errors. First, ground points can be included in  $C$  and form FPs on steep roads. We remove many of the FPs based on the comparatively flat shapes and lower locations. Second, non-ground points on a flat surface of a *target object*, such as a roof of a vehicle, are not included in  $C$  and determined as portions of ground when no ground points are assigned to the same cell. This can be minimized by adjusting the grid size appropriately,  $2.0 \times 2.0 \text{ m}^2$  in this chapter. Because vehicle widths are typically less than 2.0 m and points on the roof of a vehicle are usually assigned

to a cell with neighboring ground points, the roof points are included in  $C$  and not lost in the majority of cases. In the worst case, the roof points are completely lost, and we regard such a case as an FN if the area incorrectly determined exceeds one third of the *target object* in the  $x - y$  plane, following the definition of FNs in chapter 2.6.2. This type of error, where some non-ground points are determined as ground points, is also observed in the previous algorithms [1] and [2] when ground extraction errors arise. The FPs and FNs due to the incorrect generation of  $C$  in our experiments are all counted in Tables 2.4, 2.5, and 2.6, and demonstrate that our algorithm still outperforms the previous two algorithms for most evaluation metrics.

### 2.6.5 Impact on Tracking

In this subsection, we verify the improvement of tracking accuracy owing to the over-segmentation handling in our algorithm. To test the impact of over-segmentation handling on tracking, we built an integrated perception system to run our segmentation algorithm and a tracking algorithm in a sequential manner. As tracking inputs, parked vehicles among the *target objects* were selected due to the accurate ground truth velocities that we were able to compute. Because the parked vehicles were observed as though they were moving in the opposite direction to that of our driving platform, their relative ground truth velocities could be obtained based on our driving platform’s velocities measured by our positioning system similar to that in [21]. We computed the RMSE that determines the extent of the differences between the ground truth velocities and the estimated velocities of the parked vehicles to observe the tracking performance. As a tracking algorithm, a variant of a

Table 2.7 Impact on tracking.

Road type	Parked vehicles	Algorithm	FNs & Under seg.	Segments	Segments per vehicle	RMSE (m/s)	Gain (%)
A	1515	Proposed W/ GP	50	1525	1.04	1.77	-
		Proposed W/O GP	50	1683	1.15	2.01	11.9
		Polar grid-based	45	1640	1.12	2.25	21.3
		Mesh-based	39	1664	1.13	2.12	16.5
B	1003	Proposed W/ GP	174	884	1.07	1.64	-
		Proposed W/O GP	161	1034	1.23	1.85	11.4
		Polar grid-based	202	933	1.16	2.44	32.8
		Mesh-based	196	944	1.17	3.03	45.9

GM-PHD filter [41] showing excellent performance in multiple target tracking was chosen.

We prepared two types of segmentation results with the proposed algorithm, where over-segmentation handling was performed in only one result. All segments appeared on the parked vehicles were contained in the two results regardless of whether they were true positives or portions of over-segmentation. FNs and under-segmented objects that exceeded the typical size of vehicles,  $2.0 \times 6.0 \text{ m}^2$  in this chapter, were not included. To compare the effect of under-segmentation increased due to the faulty over-segmentation handling, We performed tracking experiments with road types A and B from our own dataset, where no under-segmentations and the most number of under-segmentations occurred, respectively, for the parked vehicles using our algorithm. The tracking results are presented in Table 2.7. Our over-segmentation handling achieves over 11% of RMSE improvements in both cases. This verifies that our over-segmentation handling obviously helps the tracking algorithm operate correctly by reducing the number of over-segmented objects. Particu-

lary, our over-segmentation handling increased only 13 under-segmentations, yet it reduced 137 over-segmentations on the type B road. This gave the RMSE improvement of 11.4%, which is similar to the result of the type A road. This implies that our over-segmentation handling is sufficiently robust against under-segmentation, and the damages due to the increased under-segmentations are sufficiently minor to be insignificant compared to the benefit of over-segmentation handling.

Note that we also carried out tracking tests with the other two segmentation algorithms, where the resulting RMSEs were higher than that of the proposed algorithm in both road types. However, it might be misleading to claim that the gaps between the tracking performances are due to our over-segmentation handling. Without over-segmentation handling, the proposed algorithm generated higher values of segments per vehicle computed by  $\text{Segments}/(\text{Parked vehicles} - (\text{FNs} + \text{Under Seg.}))$  in both road types and still achieved lower RMSEs. This means that the segments per vehicle factor is not the only factor to affect tracking performance, although it is evident that fewer segments per vehicle can result in lower RMSE when other factors are not changed. We observed that the number and locations of FN, under-segmentation, and increased segments due to over-segmentation also affected the tracking performance, which made it difficult to compare the tracking performances between different segmentation algorithms.

## 2.7 Conclusion

In this chapter, we proposed a real-time and accurate segmentation algorithm for 3D point clouds based on GP regression. The proposed algorithm combines a 2D grid and an undirected graph structure to perform clustering of non-ground points

without the ground extraction that is a possibly unnecessary step in a number of previous works. This procedure has the advantage of independence from ground extraction error as well as fast processing, which enables our algorithm to operate rapidly and stably in various urban road conditions. The segmentation accuracy is improved by GP regression, where over-segmentation occurring in every type of vehicle is handled by two types of GP models. This dramatically reduces the over-segmented objects, which subsequently contributes to accomplish higher tracking accuracy.

We also proposed new metrics that consist of OSR, precision, e-precision, USR, and recall for evaluation of our segmentation result with a large amount of data. Experiments with 21,803 *target objects* obtained from various slope conditions verify the excellence of the proposed algorithm in all metrics. In the analysis of contextual performance variance, our algorithm shows somewhat lower rates in recall and USR than the other algorithms. However, our algorithm still outperforms the competing algorithms in the other metrics, showing the highest OSR and e-precision in every scenario. In the worst case, our over-segmentation handling increased the number of under-segmentations by 13 while reducing the number of segments generated by over-segmentation by 137; this still provided better tracking accuracy by 11.4%. This indicates that the harm caused by the increased under-segmentation is minor compared with the benefit of overcoming over-segmentation. With all the contributions, our algorithm achieves real-time performance measured by 40.1 ms per frame on average.

## Chapter 3

# Pedestrian recognition based on appearance variation learning

### 3.1 Introduction

Developing a high accuracy pedestrian recognition algorithm is essential for realizing automated driving vehicles, because pedestrians are one of the most frequently appearing objects in urban driving environments. One important requirement of the recognition algorithm is to provide accurate positional information to avoid personal injury, which consistently arouses demands for 3D LIDAR deployment in automated driving vehicles. Furthermore, 3D LIDARs show greater capability to capture object shapes regardless of illumination conditions compared with other types of sensor, such as cameras or radars. Due to these advantages, 3D LIDARs have been utilized by many researchers in automated driving vehicle research [13, 19, 20, 32, 33, 42–48].

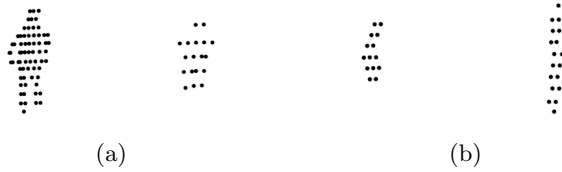


Figure 3.1 Point clouds in driving environments. (a) Point density variation of pedestrians. Distances of the left and the right are approximately 21 m and 40 m, respectively. (b) Similar appearances of a pedestrian (left) and a pole (right), both at a distance of approximately 40 m.

Recently, deep neural networks (DNNs) have been employed in a wide range of perception research areas [13, 47–53], due to their promising performance in computer vision [54]. This tendency has also appeared in automated driving vehicle research, but most of them have utilized cameras rather than 3D LIDARs. Furthermore, previous studies based on the latter have exposed a limit to recognizing pedestrians on the road [13, 47]. They tested their algorithms on 3D computer-aided design (CAD) data or noise-free and dense point cloud data that are quite different from those attained in actual driving environments. In practical driving conditions, the noise-free and dense point clouds of objects are difficult to obtain due to vehicle motion, occlusion, and/or varying point density according to distance. For example, the poles of streetlights and other road structures can raise false positives (FPs) due to a similar appearance to distant pedestrians, which is shown in Fig. 3.1. As a result, performance verification using DNN architecture on 3D point clouds from actual driving environments has been left as an unexplored area.

In this chapter, we propose a pedestrian recognition algorithm based on ap-



pearance variation learning. In driving environments, where the ego-vehicle and pedestrians are both usually moving, appearance variation information on objects is obtainable through consecutive data frames. Because appearance variation differs according to object type, utilizing the former as a cue could improve pedestrian recognition performance. Inspired by this idea, we designed a combined architecture of 3D convolutional neural network (CNN) layers and a long-short term memory (LSTM) layer [14]. First, sequential input point clouds of an object are transformed to 3D occupancy voxel grids and passed to an input layer. Appearance features of the voxel grids are then extracted by 3D CNN layers, which is followed by an LSTM layer that learns the appearance variations of the sequential voxel grids. Finally, fully connected (FC) layers perform pedestrian classification. Except for the use of the LSTM layer, our DNN architecture is similar to that of Maturana *et al.* [13], but our experimental results demonstrate that appearance variation learning due to the combination of the CNN layers and the LSTM layer clearly improves the recognition performance. This type of approach has been explored based on image data [50], but to the best of our knowledge, methodologies for the application using 3D point clouds have rarely been proposed, particularly in driving environments.

The contributions of this chapter are as follows:

- We propose a methodology to learn appearance variation of pedestrians based on a DNN architecture that employs 3D CNN layers and an LSTM layer in a sequential manner.
- We demonstrate the superiority of our DNN architecture by presenting both quantitative and qualitative results. In the quantitative study, our algorithm

was compared with state of the art algorithms based on actual driving data, and in the qualitative testing, we show that our LSTM activations gradually became different according to the object type as a result of the appearance variation learning.

The remainder of this chapter is organized as follows. Chapter 3.2 covers related works, then in chapter 3.3, the proposed DNN architecture that incorporates 3D CNN layers and an LSTM layer is described. The evaluation of the proposed DNN architecture compared to multiple baseline algorithms is provided in chapter 3.4. Finally, we conclude this chapter with a brief summary in chapter 3.5.

## 3.2 Related Work

In the literature, pedestrians have mostly been detected as one target class in multiple object classification problems [13, 19, 20, 32, 33, 42–46]. To this end, miscellaneous features, such as shape factors [32, 42, 55], shape distribution [32, 56], principal direction [32, 43, 44], point distribution in a 3D space [19, 33], reflectance [19, 42], histogram of oriented gradients (HOG) [44], spin images [32, 44, 46, 57], and global fourier histogram (GFH) descriptor [45], have been utilized in various combinations. Furthermore, in some studies, temporal features computed from consecutive data frames have been employed with the aforementioned spatial features to improve performance [20, 44, 58, 59]. These types of feature, i.e., handcrafted features, have all been devised by experts in various research areas, which implies each feature could shows comparative advantages according to application.

Over the last decade, learning algorithms for classification based on DNN archi-

tectures have rapidly emerged as another approach. In contrast to the handcrafted features, features are automatically defined and extracted using the DNN architecture by analyzing training data; this technique has shown promising performance, particularly in perception research [13, 47–54]. Among them, visual recognition studies have mostly focused on image data obtained by camera sensors as input, rather than point clouds obtained by 3D LIDARs. In particular, after Krizhevsky *et al.* [54] demonstrated the powerful performance of a well-constructed CNN architecture in the ImageNet Large Scale Visual Recognition Challenge, they have been widely utilized in various applications. In the pedestrian recognition research area, Angelova *et al.* [51] designed conventional CNNs to operate in a cascade way which achieved real-time performance. Ji *et al.* [50] devised a 3D CNN architecture to feed sequential images to an input layer for capturing temporal features of people. As a result, they achieved performance improvement on three types of human action recognition. However, because these studies only considered recognition based on 2D image data, the performance of the CNN layer on point cloud inputs cannot be approximated.

Very recently, volumetric representations of 3D shape information, such as point clouds of 3D LIDARs or depth image of 2.5D depth sensors, have been proposed. Wu *et al.* [47] suggested 3D ShapeNets that model a geometric 3D shape as a probability distribution of binary variables on a 3D voxel grid. By utilizing the 3D voxel grid with a convolutional deep belief network, they successfully performed object recognition and shape reconstruction of various types of objects. Maturana *et al.* [13] proposed VoxNet which integrates a volumetric occupancy grid with a supervised 3D CNN for object recognition with different sources of 3D data. These

volumetric representations were both appropriate to encode 3D point clouds in input forms to 3D CNN, but their experiments were mostly focused on datasets of indoor environments, which implies that their performance could be different with actual driving data.

Meanwhile, there have been another branch of DNNs that have focused on modeling the interrelationship between consecutive input data such as LSTM or gated recurrent unit (GRU) [60], which are both types of practical implementations of Recurrent Neural Networks (RNNs). Although they have been typically employed for speech recognition or text generation, they have also been utilized in other types of recognition area, such as visual recognition or maneuver anticipation. Jain *et al.* [49] fed features extracted from heterogeneous sensors to an LSTM layer, which operated well to predict five types of driver behavior. Choy *et al.* [48] proposed an extension of the standard LSTM framework they called 3D Recurrent Reconstruction Neural Network (3D-R2N2) which can perform 3D object reconstruction with both single and multi-view images. Inspired by these studies, we designed a combined architecture of 3D CNN and LSTM to improve pedestrian detection performance in actual driving environments.

In this study, we integrated 3D CNN layers and an LSTM layer to obtain better feature representation of pedestrians on road. The appearance variation of pedestrians, such as walking or running, was learned by the proposed DNN architecture based on datasets acquired from practical driving environments, and resulted in better classification performance than those in previous studies.

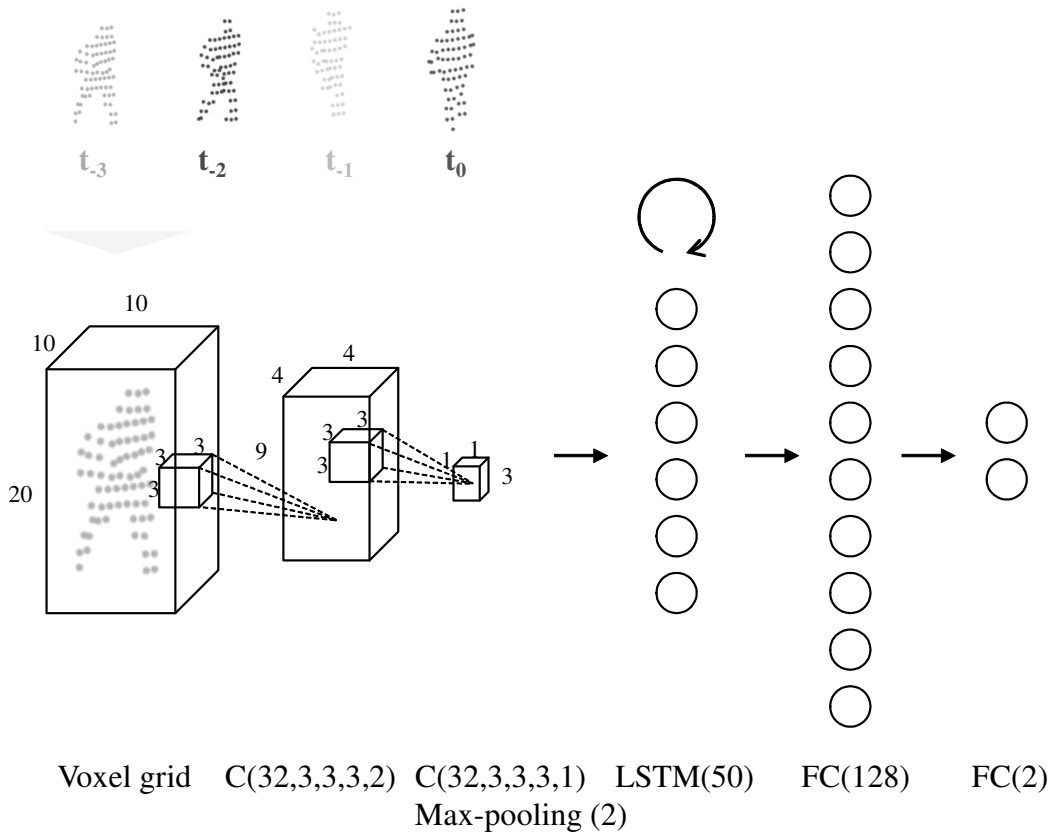


Figure 3.2 The overall architecture of the proposed DNN.

### 3.3 Appearance Variation Learning

The goal of this chapter is to recognize pedestrians on the road in actual driving conditions, where the input data are point clouds acquired by a 3D LIDAR. The key idea for the task is to make use of the appearance variation of pedestrians observed in consecutive data frames due to their motion on the road, e.g. consisting mostly of pedestrians walking and the ego-vehicle motion. Fig. 3.2 illustrates the overall architecture of the proposed DNN. First, primal input data of sequential

point clouds are transformed to 3D occupancy voxel grids. The occupancy voxel grid sequence is then passed to 3D CNN layers for capturing spatial features of pedestrian appearance in each time interval, i.e. each data frame. The activation sequence from the 3D CNN layers is then given to an RNN layer for learning temporal features in appearance variations of pedestrians as the sequence continues. Finally, FC layers yield the classification results of pedestrians and non-pedestrians by taking activations from the RNN layer.

### 3.3.1 Primal Input Data for the Proposed Architecture

We assume that segmentation of the 3D point clouds and data association over consecutive frames required to produce primal input data of the proposed DNN architecture, i.e., sequential point clouds of an object, are performed in advance. Fig. 3.3 shows the sequential point clouds of a pedestrian and a non-pedestrian that are associated over  $k$  frames, where  $k$  is four in this study. The red indicates a point cloud in the current frame  $t_0$ , the light blue is in the previous frame  $t_{-1}$ , and the other two are  $t_{-2}$  and  $t_{-3}$ , respectively. Incorrect data association that missed the same point cloud from the previous frame is also considered unless a point cloud in the current frame is valid, as shown in Fig. 3.4. Another type of incorrect data association that matches a pedestrian and non-pedestrian is not included because such cases are considered to be inappropriate for learning.

### 3.3.2 Learning Spatial Features from Appearance

Let  $p = \{p_t | t = 0, \dots, k-1\}$  be the aforementioned primal input data that represents an object sequence. Object  $p$  is associated over consecutive  $k$  data frames and  $p_t$



Figure 3.3 Sequential point clouds. Green indicates the earliest observation in  $t_{-3}$  and red indicates the current observation in  $t_0$ . (a) Pedestrian (b) Non-pedestrian



Figure 3.4 Incorrect data association. (a) Pedestrian (b) Non-pedestrian

indicates a point cloud for  $p$  in time interval  $t$ . Because the point cloud form is inappropriate for use by CNN layers, the data format needed to be changed to a 3D occupancy voxel grid of fixed size. Before voxelization of all the point clouds acquired from various positions near the ego-vehicle, they are all first translated so that their mean points become their new origins. The translated point clouds are then transformed to 3D occupancy voxel grids of fixed size  $V_W(width) \times V_D(depth) \times V_H(height)$ , where each voxel covers a volume of  $0.1 \text{ m}^3$ . If a voxel is occupied by LIDAR points, the voxel value is set to 1 and -1 otherwise, similar to the *hit grid* of [13]. These procedure allows the CNN layers to learn spatial features that are independent of the actual positions. Considering a typical volume of a single pedestrian, we set  $V_W$ ,  $V_D$ , and  $V_H$  to 10, 10, and 20, respectively. Point clouds that exceed  $1.0 \times 1.0 \times 2.0 \text{ m}^3$  are considered as obvious non-pedestrians and ignored, even though they are neighboring people.

Taking the voxel sequence as an input, CNN layers learn spatial features and

generate output activation sequence,  $x = \{x_t | t = 0, \dots, k - 1\}$ , where the dimension of  $x_t$  is the total number of output activations of the corresponding CNN layer. In this study, we employed two CNN layers  $C(32, 3, 3, 3, 2)$  and  $C(32, 3, 3, 3, 1)$ , where the first number in the parentheses is the employed filter number. The numbers from the second to the fourth positions in the parentheses are the filter size *width*, *depth*, and *height*, and the last number is the stride. After the first CNN layer, drop out regularization of factor 0.2 is implemented, whereas a max-pooling layer of  $2 \times 2 \times 2$  is applied after the second CNN layer. Drop out regularization of factor 0.3 is added to the pooling layer. As an activation function, a rectified linear unit (ReLU) [61] is utilized.

### 3.3.3 Learning Appearance Variation

The appearance variation observed over consecutive data frames can be learned by RNN layers. After receiving an activation sequence from the second CNN layer, e.g.,  $x = \{x_t | t = 0, \dots, k - 1\}$ , hidden state vector  $h_t$ , or activations from a conventional RNN layer, is computed as follows:

$$h_t = f(Wx_t + Uh_{t-1}), \quad (3.1)$$

where  $W$  and  $U$  are parameter matrices that need to be learned, and  $f$  is an activation function that is typically a logistic sigmoid function or a hyperbolic tangent function. Initial  $h_t$  is typically set to zeros. However, the conventional RNN layer has shown a limit to analyzing long sequence input, which has led to better implementation of the RNN concept, such as an LSTM layer or a GRU layer.

In this study, we chose an LSTM layer because it showed slightly better performance than a GRU layer in our experiments. In an LSTM layer, hidden state  $h_t$  is



computed as follows:

$$i_t = \sigma (W_i x_t + U_i h_{t-1} + b_i), \quad (3.2)$$

$$f_t = \sigma (W_f x_t + U_f h_{t-1} + b_f), \quad (3.3)$$

$$o_t = \sigma (W_o x_t + U_o h_{t-1} + b_o), \quad (3.4)$$

$$\tilde{c}_t = \tanh (W_c x_t + U_c h_{t-1} + b_c), \quad (3.5)$$

$$c_t = f_t \circ c_{t-1} + i_t \circ \tilde{c}_t, \quad (3.6)$$

$$h_t = o_t \circ \tanh (c_t), \quad (3.7)$$

where  $W_{(\cdot)}, U_{(\cdot)}, b_{(\cdot)}$  are parameters to be learned. After receiving an input of the current time interval  $x_t$ , the current LSTM memory cell state  $c_t$  is computed by combining  $c_{t-1}$  and a candidate memory cell state  $\tilde{c}_t$  that contains new data to be updated. The portions of  $c_{t-1}$  and  $\tilde{c}_t$  to be used are controlled by element-wise multiplications  $\circ$  with a forget gate  $f_t$  and an input gate  $i_t$ , yielding values between 0 and 1 due to logistic sigmoid function  $\sigma$ .  $h_t$  is then computed by passing  $\tanh(c_t)$  through an output gate  $o_t$ , also with values between 0 and 1.

Using Eq. (3.2) - Eq. (3.7), an LSTM layer appropriately learns the appearance variations observed over the input sequence  $x_t$  originating from the primal point cloud sequence such as in Fig. 3.3. In the next section, we demonstrate the different LSTM activations according to the input type, i.e. pedestrians and non-pedestrians, which leads to better recognition performance. The number of neurons in the LSTM layer is set to 50.

### 3.3.4 Classification

All the activations,  $h_t$ , from the LSTM layer are connected to an FC layer of 128 neurons followed by another FC layer of two neurons to make the final classification result. As an activation function, an ReLU function and a softmax function are employed in the first and second FC layers, respectively.

### 3.3.5 Data Augmentation

The occupancy distribution in the occupancy voxel grid, i.e. the input format of our DNN, can be different according to the viewpoint, even with the same point cloud. Thus, we perform data augmentation by rotating the acquired raw data point clouds by 30 degrees each along a yaw direction, which increases the number of data samples by 12 times. Considering that pedestrians rarely show behavior along a roll or pitch direction in a typical driving environment, data augmentation involved in those directions is not performed.

### 3.3.6 Implementation Detail

As a loss function, we utilized categorical cross entropy in the following form:

$$L(p, q) = - \sum_x p(x) \log(q(x)), \quad (3.8)$$

where  $p(x)$  is the probability of a ground truth label that is typically a one-hot vector and  $q(x)$  is the probability of a predicted label computed by the softmax function of the last FC layer. To optimize the loss function, the RmsProp [62] algorithm was employed; the initial learning rate was set to 0.001 and the number of epochs was set to 60. For programming, we utilized the Keras library [63] on the

Theano [64] backend. As a result, around 64,762 parameters were learned.

## **3.4 EXPERIMENTS**

In this section, we aim to prove the superiority of the proposed DNN architecture over baseline algorithms and analyze performance variation according to distance by performing a large number of experiments. We demonstrate the superiority in two ways: one is a quantitative analysis and the other is a qualitative one. In the quantitative analysis, the performance of our algorithm is compared with five state-of-the-art methods based on the KITTI dataset and our own dataset. In the qualitative analysis, we show different transitions of the LSTM activations according to different object types, which results in improved performance in the proposed algorithm.

### **3.4.1 Experimental Environment**

For evaluating the proposed DNN architecture, point cloud datasets of pedestrians and non-pedestrians associated with time sequence are required. However, because there are no appropriate publicly available datasets for our purposes, we manually collected, associated, and labeled point cloud sequences to construct one. We mounted a Velodyne HDL-64E S2 LIDAR on the roof of a vehicle and acquired 3D point cloud data by driving around the campus road at Seoul National University (SNU), when many people were walking around. On the campus road, there are many uphill and downhill roads as well as intersections and speed bumps, which helped to incorporate factors such as vehicle motion, occlusion, or varying point density (as discussed in chapter 3.1) into the dataset. Consequently, 1,670 positive

Table 3.1 Positive sample composition ratio according to distance

Dataset	Our own data							KITTI data						
Distance (m)	0-10	10-20	20-30	30-40	40-50	50-60	Total	0-10	10-20	20-30	30-40	40-50	50-60	Total
Samples	125	474	525	340	175	31	<b>1670</b>	100	238	172	336	162	36	<b>1044</b>
Percentage (%)	7.4	28.4	31.4	20.4	10.5	1.9	<b>100</b>	9.6	22.8	16.5	32.2	15.5	3.4	<b>100</b>

samples and 1,670 negative samples inside a 60 m range were collected. The left half of Table. 3.1 shows the positive sample composition ratio of our own dataset, according to distance. Beyond a 40 m range, where the point density of objects begins to decrease significantly, many objects are difficult to classify even by eye, which results in less samples than closer range.

With the constructed dataset, our algorithm and the following two algorithms were trained for quantitative evaluation:

- SF: the object recognition algorithm [42] that mainly utilizes shape factors [55] and a Support Vector Machine as a feature vector and a classifier, respectively.
- Voxnet: the object recognition algorithm based on a DNN architecture, where 3D CNN layers and FC layers without any RNN layers are utilized [13].

By testing with SF, we intended to compare the performance of the handcrafted features and the automatically extracted features, and by testing with Voxnet, we hoped to verify the performance improvement due to the employment of RNN layers, i.e. LSTM and GRU, in this study.

Furthermore, we used two types of input point clouds for testing the baseline algorithms as follows:

- *Sing* is an input point cloud for an object in a single time interval.
- *Accu* is an input point cloud for an object that has been accumulated over multiple time intervals.

*Sing* is the typical form of point cloud, such as those for  $t_0$  in Figs. 3.3 and 3.4. The other form, *Accu*, has been employed in previous studies [20, 21] to complement sparsity of distant point clouds, which can be considered as another approach to exploit temporal information of pedestrians in driving environments. Thus, we intended to compare the performance improvement by the employment of *Accu* in the baseline algorithm and that by appearance variation learning based on sequential point clouds in our algorithm. Finally, we also tested the performance of a GRU layer instead of the LSTM layer in the proposed architecture. As a result, seven methods are compared in this chapter, i.e. P.LSTM, P.GRU, P.LSTM.*Accu*, SF.*Sing*, SF.*Accu*, Voxnet.*Sing*, and Voxnet.*Accu*, where P denotes the proposed algorithm and the italicized words indicate the input point cloud formats mentioned above in the baseline algorithms. We set the sequence length of the LSTM layer to one for P.LSTM.*Accu*, and utilized *Accu* input for an additional comparison.

To train the Voxnet, an Adam optimizer was utilized because it showed better performance than the RMSProp. The number of epochs was 240 and 60 for Voxnet.*Sing* and Voxnet.*Accu*, respectively, because the sequential point clouds and *Accu* can be considered as utilizing more combinations with the same input point clouds. The initial learning rate was set to 0.001 in both cases.

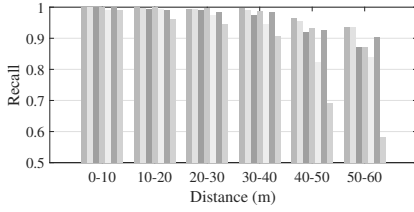
### 3.4.2 Experimental Results

For quantitative evaluation, we conducted  $k$ -fold cross validation with  $k = 10$  on all methods discussed in the previous subsection. In the training phase, training data were augmented as described in chapter 3.3.5., which resulted in 36,072 training samples in each fold. In the testing phase, classification results were determined by the voting method in Voxnet [13], where each test sample is classified in twelve directions and the voting results from the twelve classifications are utilized for the final decision. The evaluation results are shown in the upper half of Table 3.2. As can be noted from the table, our DNN architecture with the LSTM layer outperformed all of the other baseline methods except for the precision of SF.*Accu*, which demonstrates the benefit of the proposed appearance variation learning. Specifically, from the comparison of the proposed algorithm and Voxnet, we can see that employing the appearance variation learning contributed more to the improvement in accuracy than *Accu* utilization when the other conditions were the same. The employment of *Accu* also provided a performance improvement in all of the baseline algorithms, and achieved the best precision in the particular case of SF.*Accu*. This implies that the handcrafted features can also be highly improved by adjusting related factors.

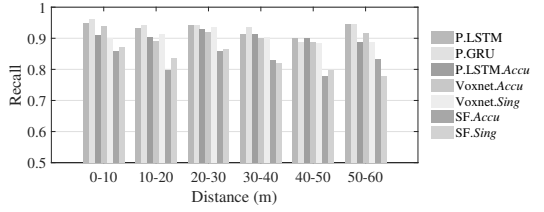
In an analysis of performance variation according to distance, as shown in Table 3.3, the superiority of the proposed architecture was intensified as it yielded better recalls in most distance ranges than the other methods. In particular, the performance gap between the proposed methods and the others was gradually enlarged beyond the 40 m range where point density significantly decreases, which implies

Table 3.2 Performance comparison

	Algorithms	P.LSTM	P.GRU	P.LSTM.Accu	Voxnet.Accu	Voxnet.Sing	SF.Accu	SF.Sing
Our own data	True Positives	1658	1653	1635	1648	1599	1635	1524
	False Positives	42	48	54	50	129	<b>28</b>	281
	Precision	0.975	0.972	0.968	0.971	0.925	<b>0.983</b>	0.844
	False Negatives	<b>12</b>	17	35	22	71	35	146
	Recall	<b>0.993</b>	0.990	0.980	0.987	0.957	0.979	0.913
KITTI data	True Positives	973	974	951	944	947	859	868
	False Positives	356	355	340	357	374	<b>93</b>	260
	Precision	0.732	0.733	0.737	0.726	0.717	<b>0.902</b>	0.770
	False Negatives	71	<b>70</b>	93	100	97	185	176
	Recall	0.932	<b>0.933</b>	0.911	0.904	0.907	0.823	0.831



(a)



(b)

Figure 3.5 Recalls of the compared architectures according to distance

that further recognition ranges can be achieved with our methods. In addition, as Fig. 3.5(a) illustrates, recalls of the proposed architectures gradually decreased beyond the 40 m range, whereas a comparatively rapid drop in performance was observed with the other baselines. This implies that more stable performance is obtained by appearance variation learning for distant objects.

To evaluate the compared algorithms with a dataset of a different environment, we utilized the KITTI tracking dataset [65]. We labeled some pedestrian/non-pedestrian samples in 21 training sequences, which amounted to 1,044 positive samples and 1,044 negative samples. The right half of Table. 3.1 shows the positive

Table 3.3 Recalls of the compared algorithms according to distance

Dataset	Our own data						KITTI data					
Distance (m)	0-10	10-20	20-30	30-40	40-50	50-60	0-10	10-20	20-30	30-40	40-50	50-60
P.LSTM	<b>1.000</b>	<b>1.000</b>	0.994	<b>0.997</b>	<b>0.966</b>	<b>0.935</b>	0.950	0.933	<b>0.942</b>	0.914	<b>0.901</b>	<b>0.944</b>
P.GRU	<b>1.000</b>	0.998	0.994	0.991	0.954	<b>0.935</b>	<b>0.960</b>	<b>0.941</b>	<b>0.942</b>	<b>0.935</b>	0.889	<b>0.944</b>
P.LSTM. <i>Accu</i>	0.910	0.903	0.930	0.914	0.901	0.889	0.910	0.903	0.930	0.914	<b>0.901</b>	0.889
Voxnet. <i>Accu</i>	<b>1.000</b>	<b>1.000</b>	<b>0.996</b>	0.988	0.931	0.871	0.940	0.891	0.919	0.902	0.889	0.917
Voxnet. <i>Sing</i>	0.992	0.994	0.975	0.947	0.823	0.839	0.900	0.912	0.936	0.905	0.883	0.889
SF. <i>Accu</i>	<b>1.000</b>	0.989	0.983	0.985	0.926	0.903	0.860	0.798	0.860	0.830	0.778	0.833
SF. <i>Sing</i>	0.992	0.962	0.947	0.906	0.691	0.581	0.870	0.836	0.866	0.821	0.796	0.778

sample composition ratio of the extracted KITTI dataset according to distance. The evaluation results are listed in the bottom half of Table. 3.2, the right half of Table. 3.3, and Fig. 3.5(b), where the overall tendency was similar to that in our own dataset. The reason why precision was more decreased than recall is probably the fact that the difference in the appearance of road structures is greater than that of people in Germany compared to Korea.

For the qualitative evaluation, we visualized activations of the first CNN layer and the LSTM layer. Fig. 3.6 illustrates a procedure in which the activations of the first CNN layer were computed from a pedestrian point cloud. The right image of Fig. 3.6(a) is an occupancy voxel grid that was produced by a point cloud depicted in the left image of Fig. 3.6(a). Convolution was then performed on the voxel grid with CNN kernels shown in Fig. 3.6(b) to generate activations of the CNN layer. Fig. 3.6(c) depicts the activations where contours of the voxel grid were extracted in the same way as conventional 2D CNN layers.



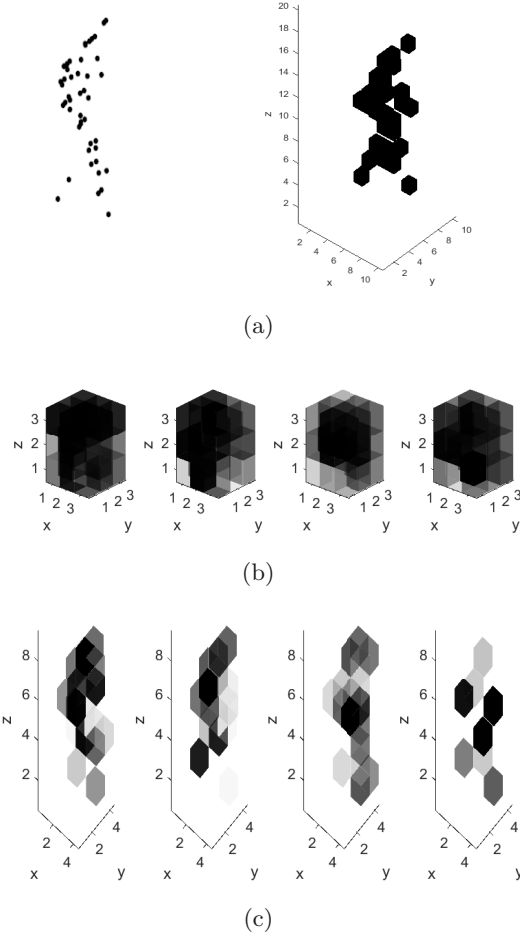


Figure 3.6 (a) A point cloud for a pedestrian and the corresponding voxel grid. (b) Weights of four 3D CNN kernels. Darker cells indicate higher values. (c) Activation of the first CNN layer computed by the four kernels in (b).

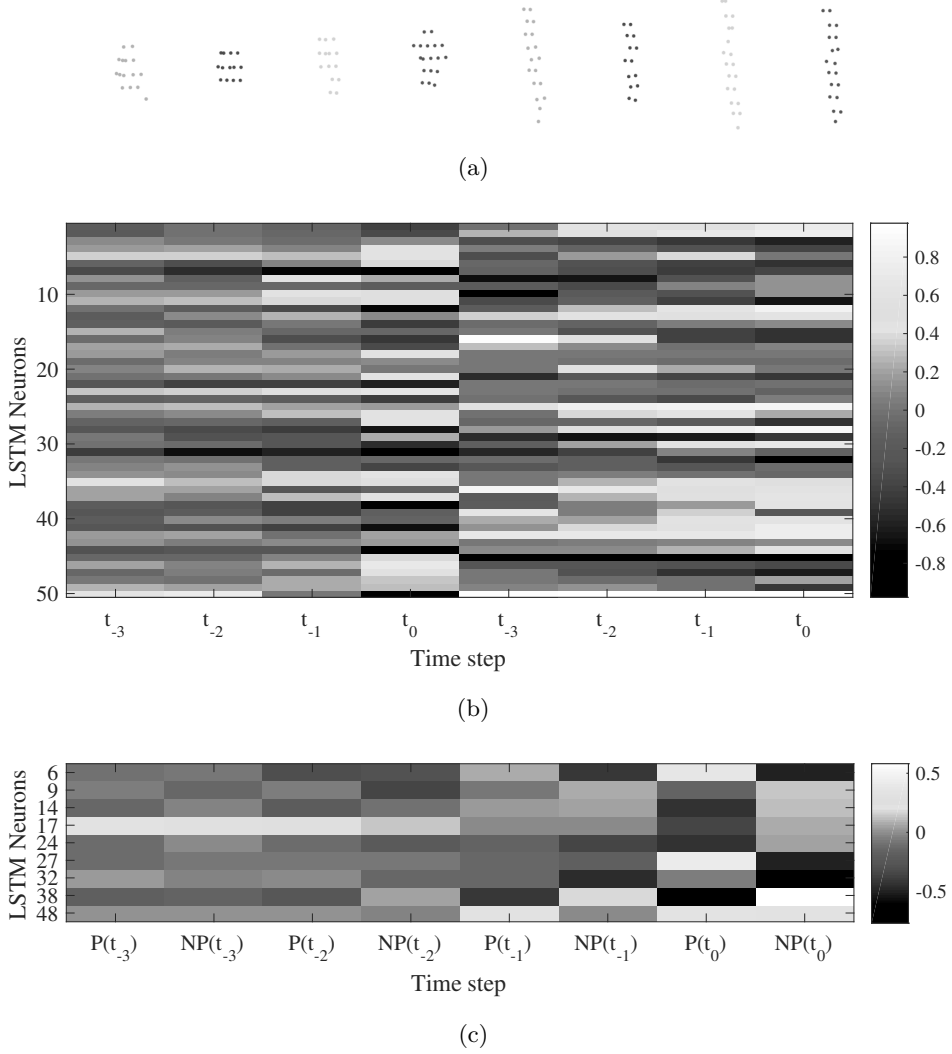


Figure 3.7 (a) Raw data sequence (b) LSTM activations (c) LSTM activation comparison according to time interval

To determine the different responses of the LSTM layer according to the sample type, i.e. a pedestrian and a non-pedestrian, we observed activation changes of the LSTM layer over time, as shown in Fig. 3.7. Fig. 3.7(a) shows the raw data sequences of a pedestrian and a non-pedestrian near the 40 m range, and the left and right sides depict sequential point clouds of the pedestrian and the non-pedestrian, respectively. Fig. 3.7(b) illustrates the activations over time where each column represents the time interval. That is to say, each column in Fig. 3.7(b) shows activation values of the corresponding point cloud for the same column in Fig. 3.7(a). Each row of Fig. 3.7(b) is an individual neuron that amounts to fifty in this study. As the figure shows, final activation results, i.e. activations in  $t_0$  of the pedestrian and the non-pedestrian, show a significant difference. To be more specific, Fig. 3.7(c) illustrates activations of some neurons that showed similar values for both the pedestrian and the non-pedestrian, denoted as P and NP on the x-axis, respectively, when appearances in the very first time interval, i.e.  $t_{-3}$ , were observed. As more input sequences were applied, i.e.  $t_{-2}$ ,  $t_{-1}$ , and  $t_0$ , the difference between activations of the pedestrian and the non-pedestrian gradually increased, whereas the final activation values in  $t_0$  were quite different. This demonstrates the result of appearance variation learning in the LSTM layer, in which an ambiguity in a distant range becomes clearer as more sequential information is given.

### 3.5 CONCLUSIONS AND FUTURE WORKS

In this study, we proposed a DNN architecture that employs 3D CNN layers and an LSTM layer in a sequential manner for pedestrian recognition in actual driving environments. A sequence of 3D point clouds acquired by a 3D LIDAR were

transformed to voxel occupancy grids of fixed sizes, then two consecutive 3D CNN layers extracted spatial features. Activations from the CNN layers underwent 3D max pooling, which was then provided to an LSTM layer. LSTM activations that incorporated appearance variation of the input data in consecutive time intervals were classified into pedestrian and non-pedestrian by two consecutive FC layers.

Experimental results demonstrate the proposed architecture outperformed the baseline algorithms without the LSTM layer in terms of both precision and recall. Additional experiments were performed with accumulated point cloud inputs where they were fed to both the proposed and baseline architecture. Although the accumulated point cloud input showed performance improvement in both precision and recall, the proposed architecture presented better performance than the others by appearance variation learning, when the other conditions were the same.

## Chapter 4

# Vehicle Recognition using a Common Appearance Captured by a 3D LIDAR and a Monocular Camera

### 4.1 Introduction

In the previous chapter, we have discussed a method to recognize pedestrians that are frequently appearing in a driving environment. In this chapter, we focus on another type of frequently appearing objects, i.e., vehicles. Detecting vehicle robustly is of great importance because in some conditions it is a higher priority than detecting pedestrians, such as in a highway driving.

A challenging issue to be considered for developing a vehicle detection algorithm is an intra-class variation. With a conventional machine learning algorithm that learns entire appearance of a target class and performs a binary classification, the same recognition performance cannot be expected for different sub-classes, such as



Figure 4.1 Different appearances of the vehicles.

sedans, buses, trucks, and SUVs. Because their appearance are actually different as shown in Fig. 4.1, the problem should be solved by a multiple class classification that requires similar number of samples for each sub-class, which results in a manual labeling time multiplied by the number of sub-classes. Thus, detecting a common part of the vehicles, such as tires and a bumper, can be efficient to solve the intra-class variation issue.

However, as we pointed out in the previous chapter, because point density of 3D LIDAR decreases as a measurement distance increases, using an only 3D LIDAR to detect tires and a bumper of a vehicle can be inappropriate. In addition, a darkness and directional edges of tires and a bumper that could provide a strong cue for detecting a vehicle is difficult to be captured by a 3D LIDAR. Therefore, we employ an additional sensor, i.e., a monocular camera, for the task in this chapter. A monocular camera provides different types of information that are mostly originated from colors, which allows extracting characteristics of the darkness and the directional edges easily compared to a 3D LIDAR.

In this chapter, we propose a vehicle recognition algorithm that uses fused in-

formation from a 3D LIDAR and a monocular camera. First, a 3D LIDAR detects candidate objects that are potentially vehicles based on their size. The pixel coordinates of the candidate objects in a gray image obtained from the monocular camera are then computed by a calibration algorithm, which is followed by detection of tires and a bumper. To detect the tires and the bumper, we utilize an observation that tires and a bumper of a vehicle show extremely low intensity values and edges of vertical directions and a horizontal direction. Lastly, Dempster-Shafer theory (DST) [15] is employed to fuse the detection results from both the 3D LIDAR and the monocular camera.

The contributions of this chapter are as follows:

- We propose a fusion algorithm that combines information from a 3D LIDAR and a monocular camera based on DST to recognize various types of vehicles.
- We propose an algorithm that detects tires and a bumper of a vehicle by using a monocular camera image.
- We present experimental results that demonstrate an improved recognition result by fusing both information from the LIDAR and the camera.

The remaining of this chapter is organized as follows. Chapter 4.2 briefly introduces related works. Chapter 4.3 then describes the proposed vehicle recognition algorithm that combines information from a 3D LIDAR and a monocular camera based on a DST framework. The performance of the proposed algorithm is provided in chapter 4.4, which is followed by chapter 4.5 that concludes this chapter with a brief summary.

## 4.2 Related Work

As one of the classical issues in automated driving research, vehicle recognition based on 3D LIDARs has been consistently studied [16, 18, 20, 32, 42–46, 66–73]. One of the major approaches is classification, where vehicles have mostly been treated as one of the multiple object classes of interest, such as pedestrians, cars, trucks, and bicycles. Thus, features and classifiers of a general purpose discussed in Chapter 3.2 [20, 32, 42–46] have been utilized for vehicle recognition. This tendency is also observed in a study focusing on only vehicle detection [16, 18]. Himmelsbach *et al.* extracted two types of features [18]. First, object level features were computed by using object intensity and volume, and histograms of point level features were computed based on the other previous studies [74, 75]. In a later study, they added motion features captured over multiple consecutive data frames to utilize a simple Naive Bayes classifier [16]. Features more oriented to recognize vehicles also have been proposed [66, 67]. Börzs *et al.* [66] made use of features extracted from a contour of a vehicle. They complemented the drawback of 2D bounding box approximation based on the Principal Component Analysis (PCA) by proposing a new box fitting method using a convex hull. Based on the new bounding box, they obtained the principal curvatures and the side view profiles as additional features. Kusenbach *et al.* [67] proposed a geometric 3D feature that can be extracted from position and intensity of points on moving objects. The point features were accumulated for multiple frames, and then utilized to build an object model that describes an entire appearance of an object class. Due to the low point density problem in a distant range, this type of approach is more appropriate to recognize the entire



appearance of an object, which cannot solve the intra-class variation problem of vehicles. With the accumulated point clouds [20,67], it is possible to learn the common appearance of vehicles, however, manual collecting of the accumulated data can be extremely time-consuming in some situations.

Another approach is a sensor fusion-based method [68–73]. In the sensor fusion-based method, features extracted from a 3D LIDAR data is combined with those from other sensors, such as cameras and radars, rather than make a final decision for vehicle recognition. DST [68,71,73] and Kalman Filter (KF) [69,70] are examples of the popular fusion frameworks. However, the sensor fusion-based approaches have rarely focused on solving the intra-class variation problem. Complementing shortcomings of each sensor for object detection, e.g., the weak capability of cameras to measure object volume, was the main purpose of sensor fusion in most studies, rather than extracting common appearances. Kim *et al.* [69] solved a timing issue that occurred due to different frame rates of sensors and processing time by using a KF. Cho *et al.* [70] devised two types of motion models to fuse features from different sensors under an Extended Kalman Filter (EKF) framework, where asynchronous data acquisition problem is solved by sequential-sensor method [76]. Chavez-Garcia *et al.* [68] employed a DST framework to solve data association problem of different sensors. Discounting factors were introduced to integrate the sensor reliability, which was also observed other studies relying on the DST framework for sensor fusion [71,73]. In the study of Vu *et al.* [71], reliability and precision of each sensor are defined to indicate sensor uncertainty, and then information with the uncertainty were integrated by a DST framework. The fusion result in the previous time step was additionally utilized in the integration. Aeberhard *et al.* [73]

primarily determined the object classes based on Gaussian Discriminant Analysis (GDA) in each sensor-level processing, which are then fused by a DST framework. To consider the different capability of each sensor for classification of a specific type of objects, they introduced a weighting factor to design the basic belief assignments (BBA). Mei *et al.* [72] did not rely on any of the aforementioned fusion frameworks to detect parked vehicles in a long range. Position and velocity information obtained from the LIDAR and the radar, respectively, were fused based on Euclidean-distance.

In this chapter, we make use of DST to fuse information acquired from a 3D LIDAR and a monocular camera, aiming to extract common appearances observed in all types of vehicles. Characteristics of tires and a bumper of a vehicle are defined by lines extracted in the tires and the bumper, where angle conditions of the lines are modeled in the DST framework to recognize vehicles.

### 4.3 Vehicle Recognition

Fig. 4.2 shows the overall processing flow of the proposed vehicle recognition algorithm of this chapter. First, a segmentation algorithm, such as the algorithm proposed in Chapter 2, yields a list of objects near the ego-vehicle. By a calibration algorithm [77], Region Of Interests (ROIs) of objects inside an FOV of our monocular camera are then passed to an image processing module in a form of pixel coordinates. The image processing module performs Connected Component Labeling (CCL) to determine that each ROI includes a connected part of tires and a bumper which is mostly extremely dark. Line extraction algorithm is implemented in ROIs where connected parts exist to find three edges on tires and

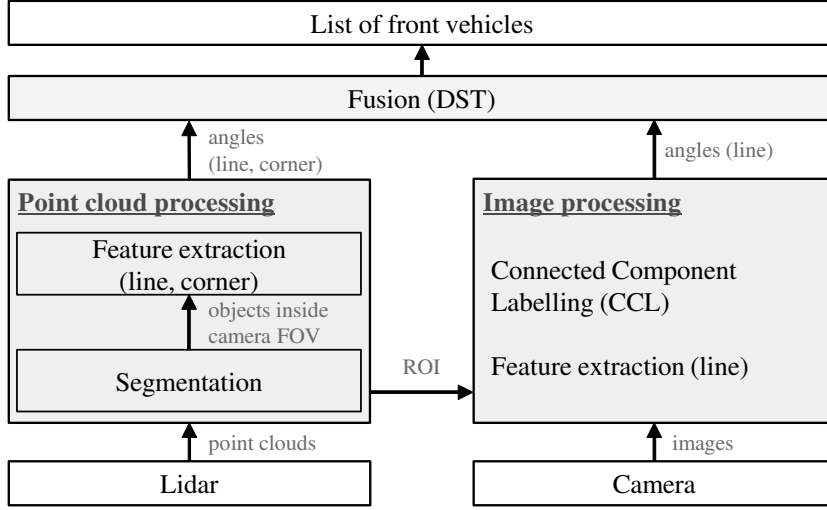


Figure 4.2 Processing flow.

the bumper. Meanwhile, line features and corner features that are usually observed on vehicles are sought in the point clouds inside the camera FOV. Finally, a DST framework fuses features obtained from the point clouds and camera images to recognize vehicles.

#### 4.3.1 Point Cloud Processing

As aforementioned, a segmentation algorithm firstly finds surrounding objects. Vehicle candidates are then determined by considering typical volume of vehicles,  $3.0 (w) \times 10.0 (d) \times 3.5 (h) \text{ m}^3$  in this chapter. The information of the vehicle candidate is passed to an image processing module in a form of  $(BottomLeft, width, height)$  that can be found from a 3D bounding box of the vehicle. Each element of the vehicle information is computed in a pixel coordinate unit by a calibration algorithm such as [77], which then generates ROIs in a camera image.



Figure 4.3 Feature extraction by shape fitting from point clouds. (a) "L" shape fitting on a sedan. The angle of the two red lines is extracted. (b) "I" shape fitting on a bus. The angle of the red line is extracted. The green points are points near the *BumperHeight* in both (a) and (b).

After the ROIs are passed to the image processing module, features that can be observed on vehicle point clouds are extracted. As discussed before, the goal of this chapter is to recognize every type of vehicle based on a common appearance, i.e., tires and a bumper, due to their similar appearance regardless of the type of a vehicle. However, performance of detecting tires with the point clouds can be limited to a short range because the small volume of tires make it difficult to be sensed in a distant range of low point density. Thus, features near a bumper are only acquired based on the LIDAR data. Points around a bumper height of a front vehicle are usually shown in two types, i.e., an "L" shape or an "I" shape [78]. The procedure of the feature extraction from the two shapes is as follows:

1. Determine a *BumperHeight* that is set to  $\tau_b + \min_z$ , where  $\tau_b$  is a threshold value and  $\min_z$  is the minimum  $z$  value of the point cloud.
2. If the maximum depth of the points near the *BumperHeight* exceeds  $\tau_d$ , a corner of the "L" shape is found. For a left side object, the leftmost point and

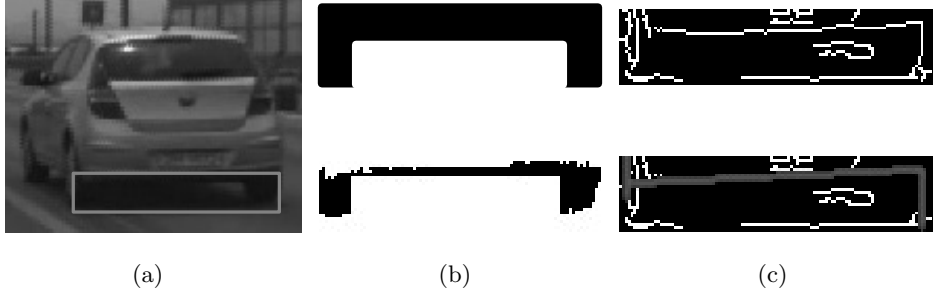


Figure 4.4 Feature extraction from a camera image. (a) A vehicle and a sliding window on the tires and a bumper. (b) "n" shape of low intensity distribution (top) and CCL result on the sliding window image (bottom). (c) A Canny edge map (top) and the extracted directional lines.

the farthest point in the depth axis are chosen as *SidePoints*. Among the points between the *SidePoints*, a point that minimizes the angle of the two lines connected from the *SidePoints* is determined as a corner point. If the object is in a right side, the rightmost point is selected instead of the leftmost point. Fig. 4.3(a) shows a vehicle and two lines connecting the corner points and the *SidePoints*. The angle of the two lines is extracted as a feature.

3. If the maximum depth is shorter than  $\tau_d$ , the "I" shape is found. Firstly a line that connects the leftmost point and the rightmost point is determined, the angle estimated by the line slope is then obtained as a feature. Fig. 4.3(b) shows a bus and a line of the "I" shape.

### 4.3.2 Image Processing

After receiving ROIs as a result of the point cloud processing, line features on tires and a bumper of a vehicle are extracted from a gray image acquired by a camera.

To reduce False Positives (FPs), low intensity values over the area, as shown in the red box of Fig. 4.4(a), are discovered first. The low intensity is another good feature of vehicles in that it is rarely affected by illumination conditions and appears in a consecutive manner over the tires and the bumper region. Thus, the region can be defined by finding an area where low intensity values are distributed in a form of an "n" shape as shown in the upper image of Fig. 4.4(b). In each ROI, a sliding window operation is implemented, as binarizing the image inside the window. Connected Component Labeling (CCL) algorithm [26] is then performed on the binarized image, where the largest area is chosen as a candidate region of tires and a bumper. If the largest CCL area forms a type of wide "n" shape, the sliding window area is selected for the line feature extraction, as shown in the lower image of Fig. 4.4(b).

From an image of tires and a bumper that is similar for every type of a vehicle, two types of lines can be extracted, i.e., a horizontal line on a bumper and a vertical line on a tire. Thus, extracting one horizontal line and two vertical lines on tires is the final task in this subsection. To transform the "n" shape of low intensity area into appropriate form for the line feature extraction, the area is converted to a Canny edge map [79]. The three directional lines are detected from the map using the Hough transform [80], and angles computed from the slopes of the lines are considered as features. The upper image of Fig. 4.4(c) illustrates the Canny edge map and the red lines in the lower image are the extracted lines.

### 4.3.3 Dempster-Shafer Theory (DST) for Information Fusion

A DST (or Evidence Theory) framework [15] models a belief of an evidence with uncertainty, which has shown an excellent performance in sensor fusion research. Let  $X$  be a universal set that includes all states of interest, or propositions. In the case of two propositions being considered, e.g.,  $X = \{a, b\}$ , the power set is then  $2^X = \{\emptyset, a, b, \{a, b\}\}$ , where the degree of the belief, or the mass, of each hypothesis is represented by a probability value in a DST framework. This is called Basic Belief Assignment (BBA):

$$m : 2^X \rightarrow [0, 1], \quad (4.1)$$

which has the following two characteristics:

$$m(\emptyset) = 0, \quad (4.2)$$

$$\sum_{A \subseteq X} m(A) = 1. \quad (4.3)$$

When beliefs are obtained from multiple sources, the degrees of the common beliefs can be derived by Dempster's rule of combination as follows:

$$m_{1,2}(\emptyset) = 0, \quad (4.4)$$

$$\begin{aligned} m_{1,2}(A) &= (m_1 \oplus m_2)(A) \\ &= \frac{1}{1 - K} \sum_{B \cap C = A \neq \emptyset} m_1(B) m_2(C), \end{aligned} \quad (4.5)$$

where

$$K = \sum_{B \cap C = \emptyset} m_1(B) m_2(C). \quad (4.6)$$

K measures the extent of conflicts between the mass sets of the different sources.

Let  $\theta_L$  be an angle computed from the lines of the "I" shape or the "L" shape in the point cloud processing module, and  $\theta_C = \{\theta_{C_1}, \theta_{C_2}, \theta_{C_3}\}$  be a set of angles computed from the directional lines in the image processing module, where  $\theta_{C_1}$  is the horizontal line. These angles are utilized for the mass assignment in each source, i.e., the LIDAR and the camera. As the goal of this chapter is to recognize vehicles, a universal set of two propositions,  $X = \{v, nv\}$ , is defined where  $v$  and  $nv$  denote a vehicle and a non-vehicle, respectively.

In the case of LIDAR beliefs, masses  $m_L(A)$  are designed as follows:

$$m_L(A) = \begin{cases} m_L(\{v\}) & = & \alpha_L f_L(\theta_L) \\ m_L(\{nv\}) & = & \alpha_L (1 - f_L(\theta_L)) \\ m_L(\{v, nv\}) & = & 1 - \alpha_L \end{cases} , \quad (4.7)$$

where  $f_L(\theta_L)$  is:

$$f_L(\theta_L) = \exp(-\lambda_L |\theta_L - \theta_{L_S}|) . \quad (4.8)$$

$\theta_{L_S}$  is an ideal angle that can be computed from the "I" shape and the "L" shape of a vehicle. Thus,  $\theta_{L_S}$  becomes  $0^\circ$  or  $90^\circ$  according to the shape, which forces an object of which  $\theta_L$  is quite different from  $\theta_{L_S}$  to get a low mass.  $\lambda_L$  is introduced to control the effect of  $|\theta_L - \theta_{L_S}|$  on a mass. Lastly,  $\alpha_L \in [0, 1]$  is an evidence discounting factor as in [68] to represent a degree of a belief that cannot be assigned to both  $\{v\}$  and  $\{nv\}$ .

In the case of camera beliefs, masses  $m_C(A)$  are designed as follows:

$$m_C(A) = \begin{cases} m_C(\{v\}) & = & \alpha_C f_C(\theta_C) \\ m_C(\{nv\}) & = & \alpha_C (1 - f_C(\theta_C)) \\ m_C(\{v, nv\}) & = & 1 - \alpha_C \end{cases} , \quad (4.9)$$



where  $f_C(\theta_C)$  is:

$$f_C(\theta_C) = \exp(-\lambda_{C_1} |\theta_{C_1} - \theta_{C_H}|) \exp(-\lambda_{C_2} |\theta_{C_2} - \theta_{C_V}|) \\ \exp(-\lambda_{C_3} |\theta_{C_3} - \theta_{C_V}|). \quad (4.10)$$

The meaning of parameters are similar to the case of the LIDAR. The ideal  $\theta_{C_H}$  and  $\theta_{C_V}$  are also  $0^\circ$  and  $90^\circ$ , respectively.

The two mass sets above are combined according to Eq. (4.4) - Eq. (4.6). When the joint mass  $m_{L,C}(v)$  is greater than  $m_{L,C}(nv)$ , the object is considered as a vehicle.

## 4.4 Experiments

The proposed vehicle recognition algorithm is tested with a dataset acquired from a campus road at SNU, i.e., a type of an urban driving environment, by using a Velodyne HDL-64E S2 LDIAR and a monocular camera of a  $640 \times 480$  resolution. In contrast to a highway where only vehicles appear on a comparatively flat road, other types of objects, such as pedestrians and road structures that potentially make FPs, are faced on irregular slopes in the urban driving environment. In addition, a speed bump, on which data fusion cannot be implemented well because a predefined calibration setting is invalid, allows an evaluation in a practical driving environment.

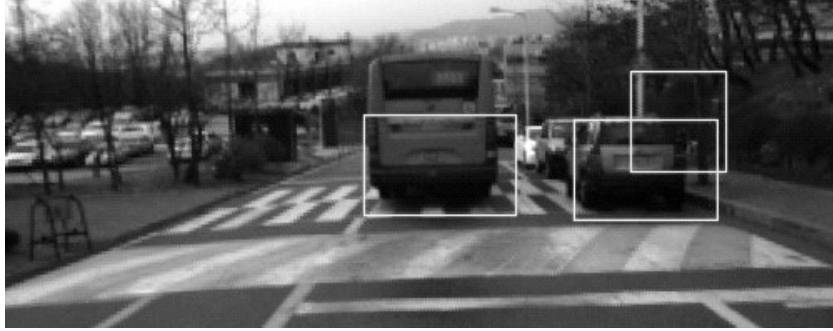
The parameters in the DST framework are set as follows.  $\alpha_L = \alpha_C = 0.9$ ,  $\lambda_L = 0.01$ ,  $\lambda_{C_1} = 0.005$ , and  $\lambda_{C_2} = 0.001$ . In addition, though the ideal angles of  $\theta_{L_S}$  is  $90^\circ$  in the case of the "L" shape, it is set to  $95^\circ$ , considering that  $\theta_L$  is

hardly a right angle due to sensor noise even if the corner of a vehicle is perfect right angle. Due to the similar reason,  $\theta_{C_V}$  is set to  $80^\circ$ .

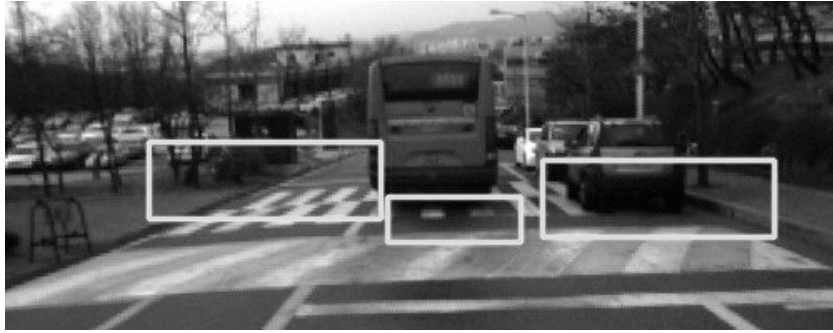
Fig. 4.5 illustrates a vehicle recognition results. In Fig. 4.5(a), light blue rectangles are the recognition results by only the point cloud processing with the line feature extraction, where the results of only the image processing are shown in Fig. 4.5(b) with green rectangles. Finally, Fig. 4.5(c) shows the fusion result in which FPs of Fig. 4.5(a) and Fig. 4.5(b) are filtered out by the DST framework. The yellow rectangles indicate the recognized vehicles with distances measured by the LIDAR.

For quantitative evaluation, vehicles in consecutive 400 image frames are labeled manually as ground truth in which vehicles that are only partially observed due to occlusion or beyond 70 m range are not included. Table 4.1 is the evaluation result on three types of vehicles, i.e., a sedan, an SUV, and a bus. The overall recall and precision are 0.890 and 0.907, respectively, where the recalls according to the vehicle types are 0.899, 0.87, and 0.85, respectively. Recalls of the different vehicle types are similar, as expected. The reason of False Negative (FN) generation can be explained as the following. First, some vehicles of black colors were not detected by the LIDAR due to their low reflection. Next, directional lines were not extracted well from an ROI of a black vehicle when the vehicle was close and in a right adjacent lane, because the intensity over the nearly entire region of the ROI was extremely low. The last reason is a distortion of the calibration setting near a speed bump. Particularly, incorrect ROIs were generated in a region of the adjacent lane.

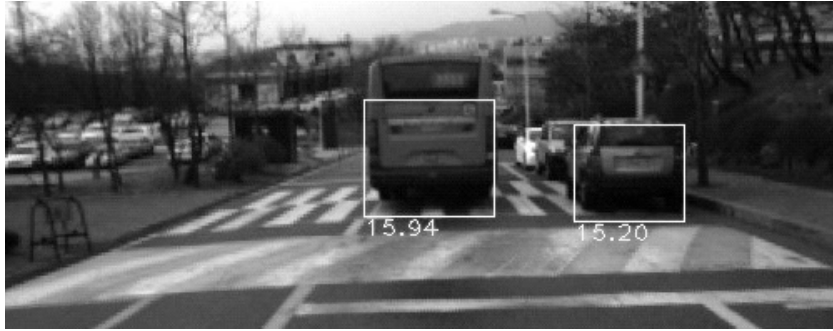
In the case of FPs, most FPs were generated from ROIs in which the three directional lines were observed due to a contour formed by adjacent scene of another



(a)



(b)



(c)

Figure 4.5 Comparison of the vehicle recognition results. (a) The result of the point cloud processing with the line feature extraction. (b) The result of the image processing obtained by a sliding window operation over the entire image. (c) The result of the information fusion by the DST framework.



Figure 4.6 Example of FPs. ROIs where the directional lines are observed due to the contour between adjacent scene mostly generated FPs.

color as shown in Fig. 4.6.

## 4.5 Conclusion

In this chapter, we have proposed a vehicle recognition algorithm that utilizes fused information acquired by a 3D LIDAR and a monocular camera. Particularly, an employment of a common appearance of vehicles, i.e., low intensity and directional lines on tires and a bumper, is the main approach for the task. In the proposed algorithm, ROIs in an image frame are firstly determined by a point cloud processing which performs segmentation and line feature extraction. Three directional lines on tires and a bumper are then extracted in each ROI by an image processing module, which is followed by a DST framework. The DST framework integrates the angle information computed from the extracted lines in both processing modules to determine vehicle presence. Experimental results demonstrate the proposed algorithm improves the vehicle recognition performance compared to the case without the information fusion. In the quantitative evaluation result, the recognition

Table 4.1 Quantitative evaluation.

Vehicle type	Bus	Sedan	SUV	Total
Samples	407	146	20	573
True Positives	366	127	17	510
False Negatives	41	19	3	63
Recall	<b>0.899</b>	<b>0.870</b>	<b>0.850</b>	<b>0.890</b>
False Positives	-	-	-	52
Precision	-	-	-	<b>0.907</b>

performance are comparatively low in the case of a sedan, compared to the other types of vehicles, such as a bus and an SUV. The reasons were threefold. Except for the low reflection problem of a black sedan, it is expected that the other two problems can be solved by improving line extraction algorithm and widening the ROI computed by a the calibration algorithm. to the employment of a common appearance of vehicles in the proposed algorithm.

## Chapter 5

# Conclusion

In this dissertation, we have proposed three perception algorithms that can be used in automated driving vehicles. All of the algorithms were based on utilizing point clouds as input, since we have inspired by the benefits of a 3D LIDAR. In chapter 2, we proposed a real-time and accurate segmentation algorithm. By skipping a ground extraction procedure that most previous works have performed, the proposed algorithm achieved real-time performance. At the same time, segmentation accuracy was also improved by reducing over-segmentation based on GP regression, which subsequently made better performance of a tracking algorithm. In chapter 3, we proposed a pedestrian recognition algorithm which learns appearance variation based on a DNN architecture that employs 3D CNN layers and an LSTM layer in a sequential manner. By learning appearance variation, the proposed DNN architecture accomplished better precision and recall simultaneously, which outperformed baseline DNN architectures. In addition, by addressing performance analysis ac-

cording to distance in actual driving environment, the study provided a criterion to determine a valid recognition range of a DNN architecture in which input data were point clouds. In chapter 4, we proposed a vehicle recognition algorithm, a main purpose of which was to achieve equal performance over various types of vehicles. For the purpose, the proposed algorithm made use of features extracted from a common appearance of vehicles, i.e., tires and a bumper. To complement a shortcoming of the LIDAR that the common appearance cannot be sensed perfectly in a distant range, a monocular camera was additionally used. Features obtained by both sensors were then combined by a DST framework. Experimental results demonstrated the information fusion actually improved perception performance, however, limits to achieving the equal performance over various vehicle types also have been shown.

# Bibliography

- [1] B. Douillard, J. Underwood, N. Kuntz, V. Vlaskine, A. Quadros, P. Morton, and A. Frenkel, “On the segmentation of 3d lidar point clouds,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 2798–2805.
- [2] M. Himmelsbach, F. v Hundelshausen, and H. Wuensche, “Fast segmentation of 3d point clouds for ground vehicles,” in *Intelligent Vehicles Symposium (IV), 2010 IEEE*, June 2010, pp. 560–565.
- [3] D.-K. Kye, M. Shin, G.-P. Gwon, S.-T. Choi, I.-S. Yoo, B.-K. Seo, S. Lee, H. Lee, Y. Lee, G. Oh, and S.-W. Kim, “Development of an autonomous vehicle : Baby in car,” in *International Conference on Electronics, Information and Communication (ICEIC 2015)*, January 2015.
- [4] G.-P. Gwon, E.-D. Lee, S.-N. Kang, M.-E. Choi, M.-O. Shin, S.-T. Choi, I.-S. Yoo, B.-K. Seo, D.-S. Baek, and S.-W. Kim, “A multi-ugv testbed for autonomous vehicle cooperative driving,” in *International Conference on Electronics, Information and Communication (ICEIC 2015)*, January 2015.



- [5] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*. Springer, 2009.
- [6] S.-W. Kim, G.-P. Gwon, S.-T. Choi, S.-N. Kang, M.-O. Shin, I.-S. Yoo, E.-D. Lee, and S.-W. Seo, “Multiple vehicle driving control for traffic flow efficiency,” in *IEEE Intelligent Vehicles Symposium (IV’12)*, June 2012.
- [7] G. P. Gwon, W. S. Hur, S. W. Kim, and S. W. Seo, “Generation of a precise and efficient lane-level road map for intelligent vehicle systems,” *IEEE Transactions on Vehicular Technology*, vol. PP, no. 99, pp. 1–1, 2016.
- [8] J. Levinson and S. Thrun, “Robust vehicle localization in urban environments using probabilistic maps,” in *2010 IEEE International Conference on Robotics and Automation*, May 2010, pp. 4372–4378.
- [9] S. Kang, I. Yoo, M. Shin, and S. Seo, “Accurate inter-vehicle distance measurement based on monocular camera and line laser,” *IEICE Electronics Express*, vol. 11, no. 9, pp. 20 130 932–20 130 932, 2014.
- [10] M. Shin, M.-E. Choi, G.-P. Gwon, S.-T. Choi, I.-S. Yoo, B. Seo, G. Oh, D. Baek, and S.-W. Kim, “A track boundary detection method by utilizing temporary local coordinate,” in *International Conference on Electronics, Information and Communication (ICEIC 2015)*, January 2015.
- [11] D. K. Kye, S. W. Kim, and S. W. Seo, “Decision making for automated driving at unsignalized intersection,” in *2015 15th International Conference on Control, Automation and Systems (ICCAS)*, Oct 2015, pp. 522–525.

- [12] C. E. Rasmussen and C. K. I. Williams, *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2006.
- [13] D. Maturana and S. Scherer, “VoxNet: A 3D Convolutional Neural Network for Real-Time Object Recognition,” in *IROS*, 2015.
- [14] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997. [Online]. Available: <http://dx.doi.org/10.1162/neco.1997.9.8.1735>
- [15] P. Smets and R. Kennes, *The Transferable Belief Model*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 693–736.
- [16] M. Himmelsbach and H.-J. Wuensche, “Tracking and classification of arbitrary objects with bottom-up/top-down detection,” in *Intelligent Vehicles Symposium (IV), 2012 IEEE*, June 2012, pp. 577–582.
- [17] J. Cheng, Z. Xiang, T. Cao, and J. Liu, “Robust vehicle detection using 3d lidar under complex urban environment,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, May 2014, pp. 691–696.
- [18] M. Himmelsbach, T. Luettel, and H. Wuensche, “Real-time object classification in 3d point clouds using point feature histograms,” in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, Oct 2009, pp. 994–1000.

- [19] K. Kidono, T. Miyasaka, A. Watanabe, T. Naito, and J. Miura, “Pedestrian recognition using high-definition lidar,” in *Intelligent Vehicles Symposium (IV)*, 2011 *IEEE*, June 2011, pp. 405–410.
- [20] A. Teichman, J. Levinson, and S. Thrun, “Towards 3d object recognition via classification of arbitrary object tracks,” in *Robotics and Automation (ICRA)*, 2011 *IEEE International Conference on*, May 2011, pp. 4034–4041.
- [21] D. Held, J. Levinson, S. Thrun, and S. Savarese, “Combining 3d shape, color, and motion for robust anytime tracking,” in *Proceedings of Robotics: Science and Systems*, Berkeley, USA, July 2014.
- [22] Y. Zhou, D. Wang, X. Xie, Y. Ren, G. Li, Y. Deng, and Z. Wang, “A fast and accurate segmentation method for ordered lidar point cloud of large-scale scenes,” *Geoscience and Remote Sensing Letters, IEEE*, vol. 11, no. 11, pp. 1981–1985, Nov 2014.
- [23] T. Chen, B. Dai, R. Wang, and D. Liu, “Gaussian-process-based real-time ground segmentation for autonomous land vehicles,” *Journal of Intelligent and Robotic Systems*, vol. 76, no. 3-4, pp. 563 – 582, December 2014.
- [24] B. Yang and Z. Dong, “A shape-based segmentation method for mobile laser scanning point clouds,” *{ISPRS} Journal of Photogrammetry and Remote Sensing*, vol. 81, pp. 19 – 30, 2013. [Online]. Available: <http://www.sciencedirect.com/science/article/pii/S0924271613001020>

- [25] Y. Yu, J. Li, H. Guan, and C. Wang, “Automated extraction of urban road facilities using mobile laser scanning data,” *Intelligent Transportation Systems, IEEE Transactions on*, vol. 16, no. 4, pp. 2167–2181, Aug 2015.
- [26] G. Stockman and L. G. Shapiro, *Computer Vision*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.
- [27] A. Das, J. Servos, and S. L. Waslander, “3d scan registration using the normal distributions transform with ground segmentation and point cloud clustering,” in *2013 IEEE International Conference on Robotics and Automation*, May 2013, pp. 2207–2212.
- [28] S. K. Reddy and P. K. Pal, “Segmentation of point cloud from a 3d lidar using range difference between neighbouring beams,” in *Proceedings of the 2015 Conference on Advances In Robotics*, ser. AIR ’15. New York, NY, USA: ACM, 2015, pp. 55:1–55:6. [Online]. Available: <http://doi.acm.org/10.1145/2783449.2783504>
- [29] —, “Computing an unevenness field from 3d laser range data to obtain traversable region around a mobile robot,” *Robot. Auton. Syst.*, vol. 84, no. C, pp. 48–63, Oct. 2016. [Online]. Available: <http://dx.doi.org/10.1016/j.robot.2016.07.002>
- [30] H. Yin, X. Yang, and C. He, “Spherical coordinates based methods of ground extraction and objects segmentation using 3-d lidar sensor,” *IEEE Intelligent Transportation Systems Magazine*, vol. 8, no. 1, pp. 61–68, Spring 2016.

- [31] F. Moosmann, O. Pink, and C. Stiller, “Segmentation of 3d lidar data in non-flat urban environments using a local convexity criterion,” in *Intelligent Vehicles Symposium, 2009 IEEE*, June 2009, pp. 215–220.
- [32] D. Wang, I. Posner, and P. Newman, “What could move? finding cars, pedestrians and bicyclists in 3d laser data,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, May 2012, pp. 4038–4044.
- [33] L. Spinello, K. O. Arras, R. Triebel, and R. Siegwart, “A layered approach to people detection in 3d range data.” in *Proc. of The AAAI Conference on Artificial Intelligence: Physically Grounded AI Track (AAAI)*, 2010.
- [34] M. Smith, I. Posner, and P. Newman, “Adaptive compression for 3d laser data,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 914 – 935, June 2011.
- [35] S. Vasudevan, F. Ramos, E. Nettleton, and H. Durrant-Whyte, “Gaussian process modeling of large-scale terrain,” *Journal of Field Robotics*, vol. 26, no. 10, pp. 812–840, 2009. [Online]. Available: <http://dx.doi.org/10.1002/rob.20309>
- [36] T. Chen, B. Dai, D. Liu, J. Song, and Z. Liu, “Velodyne-based curb detection up to 50 meters away,” in *Intelligent Vehicles Symposium (IV), 2015 IEEE*, June 2015, pp. 241–248.
- [37] K. Kim, D. Lee, and I. Essa, “Gaussian process regression flow for analysis of motion trajectories,” in *Computer Vision (ICCV), 2011 IEEE International Conference on*, Nov 2011, pp. 1164–1171.

- [38] P. Trautman and A. Krause, “Unfreezing the robot: Navigation in dense, interacting crowds,” in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, Oct 2010, pp. 797–803.
- [39] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *International Journal of Robotics Research (IJRR)*, 2013.
- [40] R. B. Rusu and S. Cousins, “3d is here: Point cloud library (pcl),” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, May 2011, pp. 1–4.
- [41] M. Choi and S. Seo, “Robust multi-target tracking scheme based on gaussian mixture probability hypothesis density filter,” *Vehicular Technology, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2015.
- [42] D. Z. Wang and I. Posner, “Voting for voting in online point cloud object detection,” in *Proceedings of Robotics: Science and Systems*, Rome, Italy, July 2015.
- [43] A. Azim and O. Aycard, “Layer-based supervised classification of moving objects in outdoor dynamic environment using 3d laser scanner,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 1408–1414.
- [44] M. Delp, N. Nagasaka, N. Kamata, and M. R. James, “Classifying and passing 3d obstacles for autonomous driving,” in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sept 2015, pp. 1240–1247.

- [45] T. Chen, B. Dai, D. Liu, and J. Song, “Performance of global descriptors for velodyne-based urban object recognition,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 667–673.
- [46] J. Behley, V. Steinhage, and A. B. Cremers, “Laser-based segment classification using a mixture of bag-of-words,” in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nov 2013, pp. 4195–4200.
- [47] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao, “3d shapenets: A deep representation for volumetric shapes,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 1912–1920.
- [48] C. B. Choy, D. Xu, J. Gwak, K. Chen, and S. Savarese, “3d-r2n2: A unified approach for single and multi-view 3d object reconstruction,” in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2016.
- [49] A. Jain, H. S. Koppula, S. Soh, B. Raghavan, A. Singh, and A. Saxena, “Brain4cars: Car that knows before you do via sensory-fusion deep learning architecture,” *CoRR*, vol. abs/1601.00740, 2016. [Online]. Available: <http://arxiv.org/abs/1601.00740>
- [50] S. Ji, W. Xu, M. Yang, and K. Yu, “3d convolutional neural networks for human action recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 35, no. 1, pp. 221–231, Jan 2013.

- [51] A. Angelova, A. Krizhevsky, V. Vanhoucke, A. Ogale, and D. Ferguson, “Real-time pedestrian detection with deep network cascades,” in *Proceedings of BMVC 2015*, 2015.
- [52] B. Huval, T. Wang, S. Tandon, J. Kiske, W. Song, J. Pazhayampallil, M. Andriluka, P. Rajpurkar, T. Migimatsu, R. Cheng-Yue, F. Mujica, A. Coates, and A. Y. Ng, “An empirical evaluation of deep learning on highway driving,” *CoRR*, vol. abs/1504.01716, 2015. [Online]. Available: <http://arxiv.org/abs/1504.01716>
- [53] V. Badrinarayanan, A. Kendall, and R. Cipolla, “Segnet: A deep convolutional encoder-decoder architecture for image segmentation,” *arXiv preprint arXiv:1511.00561*, 2015.
- [54] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “Imagenet classification with deep convolutional neural networks,” in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: <http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- [55] C.-F. Westin, S. Peled, H. Gudbjartsson, R. Kikinis, and F. A. Jolesz, “Geometrical diffusion measures for MRI from tensor basis analysis,” in *ISMRM '97*, Vancouver Canada, April 1997, p. 1742.
- [56] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin, “Shape distributions,” *ACM Trans. Graph.*, vol. 21, no. 4, pp. 807–832, Oct. 2002. [Online].



Available: <http://doi.acm.org/10.1145/571647.571648>

- [57] A. E. Johnson and M. Hebert, “Using spin images for efficient object recognition in cluttered 3d scenes,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 21, no. 5, pp. 433–449, May 1999. [Online]. Available: <http://dx.doi.org/10.1109/34.765655>
- [58] W. Xiao, B. Vallet, K. Schindler, and N. Paparoditis, “Simultaneous detection and tracking of pedestrian from panoramic laser scanning data,” *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. III-3, pp. 295–302, 2016. [Online]. Available: <http://www.isprs-ann-photogramm-remote-sens-spatial-inf-sci.net/III-3/295/2016/>
- [59] Y. Yi, Y. Guang, Z. Hao, F. Meng-yin, and W. Mei-ling, “Moving object detection under dynamic background in 3d range data,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 394–399.
- [60] J. Chung, Ç. Gülçehre, K. Cho, and Y. Bengio, “Empirical evaluation of gated recurrent neural networks on sequence modeling,” *CoRR*, vol. abs/1412.3555, 2014. [Online]. Available: <http://arxiv.org/abs/1412.3555>
- [61] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, J. Furnkranz and T. Joachims, Eds. Omnipress, 2010, pp. 807–814. [Online]. Available: <http://www.icml2010.org/papers/432.pdf>

- [62] Y. N. Dauphin, H. de Vries, J. Chung, and Y. Bengio, “Rmsprop and equilibrated adaptive learning rates for non-convex optimization,” *CoRR*, vol. abs/1502.04390, 2015. [Online]. Available: <http://arxiv.org/abs/1502.04390>
- [63] F. Chollet, “Keras,” <https://github.com/fchollet/keras>, 2015.
- [64] Theano Development Team, “Theano: A Python framework for fast computation of mathematical expressions,” *arXiv e-prints*, vol. abs/1605.02688, May 2016. [Online]. Available: <http://arxiv.org/abs/1605.02688>
- [65] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.
- [66] A. Börcs, B. Nagy, M. Baticz, and C. Benedek, *A Model-Based Approach for Fast Vehicle Detection in Continuously Streamed Urban LIDAR Point Clouds*. Cham: Springer International Publishing, 2015, pp. 413–425. [Online]. Available: [http://dx.doi.org/10.1007/978-3-319-16628-5\\_30](http://dx.doi.org/10.1007/978-3-319-16628-5_30)
- [67] M. Kusenbach, M. Himmelsbach, and H. J. Wuensche, “A new geometric 3d lidar feature for model creation and classification of moving objects,” in *2016 IEEE Intelligent Vehicles Symposium (IV)*, June 2016, pp. 272–278.
- [68] R. O. Chavez-Garcia and O. Aycard, “Multiple sensor fusion and classification for moving object detection and tracking,” *IEEE Transactions on Intelligent Transportation Systems*, vol. 17, no. 2, pp. 525–534, Feb 2016.
- [69] S. Kim, H. Kim, W. Yoo, and K. Huh, “Sensor fusion algorithm design in detecting vehicles using laser scanner and stereo vision,” *IEEE Transactions*

on *Intelligent Transportation Systems*, vol. 17, no. 4, pp. 1072–1084, April 2016.

- [70] H. Cho, Y. W. Seo, B. V. K. V. Kumar, and R. R. Rajkumar, “A multi-sensor fusion system for moving object detection and tracking in urban driving environments,” in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, May 2014, pp. 1836–1843.
- [71] T. D. Vu, O. Aycard, and F. Tango, “Object perception for intelligent vehicle applications: A multi-sensor fusion approach,” in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, June 2014, pp. 774–780.
- [72] X. Mei, N. Nagasaka, B. Okumura, and D. Prokhorov, “Detection and motion planning for roadside parked vehicles at long distance,” in *2015 IEEE Intelligent Vehicles Symposium (IV)*, June 2015, pp. 412–418.
- [73] M. Aeberhard and T. Bertram, “Object classification in a high-level sensor data fusion architecture for advanced driver assistance systems,” in *2015 IEEE 18th International Conference on Intelligent Transportation Systems*, Sept 2015, pp. 416–422.
- [74] J.-F. Lalonde, N. Vandapel, D. Huber, and M. Hebert, “Natural terrain classification using three-dimensional ladar data for ground robot mobility,” *Journal of Field Robotics*, vol. 23, no. 10, pp. 839 – 861, November 2006.
- [75] D. Anguelov, B. Taskarf, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, and A. Ng, “Discriminative learning of markov random fields for segmentation of 3d scan data,” in *2005 IEEE Computer Society Conference on Computer*

*Vision and Pattern Recognition (CVPR'05)*, vol. 2, June 2005, pp. 169–176  
vol. 2.

- [76] H. Durrant-Whyte and T. C. Henderson, *Multisensor Data Fusion*. Cham: Springer International Publishing, 2016, pp. 867–896.
- [77] G. Pandey, J. R. McBride, S. Savarese, and R. M. Eustice, “Automatic targetless extrinsic calibration of a 3d lidar and camera by maximizing mutual information,” in *Proceedings of the Twenty-Sixth AAAI Conference on Artificial Intelligence*, ser. AAAI’12. AAAI Press, 2012, pp. 2053–2059. [Online]. Available: <http://dl.acm.org/citation.cfm?id=2900929.2901018>
- [78] C. Mertz, L. E. Navarro-Serment, R. MacLachlan, P. Rybski, A. Steinfeld, A. Supp&#x00e9;, C. Urmson, N. Vandapel, M. Hebert, C. Thorpe, D. Duggins, and J. Gowdy, “Moving object detection with laser scanners,” *J. Field Robot.*, vol. 30, no. 1, pp. 17–43, Jan. 2013. [Online]. Available: <http://dx.doi.org/10.1002/rob.21430>
- [79] J. Canny, “A computational approach to edge detection,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 8, no. 6, pp. 679–698, Jun. 1986. [Online]. Available: <http://dx.doi.org/10.1109/TPAMI.1986.4767851>
- [80] J. Matas, C. Galambos, and J. Kittler, “Robust detection of lines using the progressive probabilistic hough transform,” *Comput. Vis. Image Underst.*, vol. 78, no. 1, pp. 119–137, Apr. 2000. [Online]. Available: <http://dx.doi.org/10.1006/cviu.1999.0831>



## 국문초록

3D LIDAR는 자율 주행 자동차용 인식 센서들 중, 가장 높은 정확도를 갖는 위치좌표 형태로 주변 물체들의 표면 정보를 제공한다. 스스로 레이저를 발사하고 그것이 주변 물체들로부터 반사되어 돌아오는 것을 수신함으로써 거리를 측정하는 active sensor 이기 때문에 아침, 밤 등 시간에 따라 변화하는 조명 환경에 영향을 받지 않고 안정적인 거리 정보를 제공한다는 장점이 있는데, 실 사용 예로, 상기 고정밀 거리 측정 정보는 그 자체만으로 충돌 회피에 유용하게 이용될 수 있으며, 상기 3D 표면 정보는 물체 분류 (classification)를 위한 feature로서 활용될 수 있다. 본 학위 논문에서는 상기의 3D LIDAR의 장점을 활용하는 3개의 자율 주행 자동차용 물체 인식 알고리즘을 제안한다.

3D LIDAR를 인식 센서로 활용하기 위해 가장 먼저 수행되어야 하는 과정은 segmentation이다. Segmentation은 stream 형태로 수신되는 LIDAR 데이터를 다수의 포인트 그룹으로 만들어주는데, 이 때 각각의 포인트 그룹은 센서 인근의 개별 물체 표면 형상을 이루게 된다. 2장에서는 상기 segmentation을 실시간, 고정밀로 수행하는 알고리즘을 제안한다. 특히, 하나의 물체를 다수의 조각으로 쪼개어 False Positive 에러를 증가시키는 over-segmentation 문제를 해결하기 위해 Gaussian Process 회귀가 이용된다.

상기 segmentation 결과는 분류와 같은 다른 인식 알고리즘의 입력 데이터로 이용될 수 있다. 분류 알고리즘의 경우, 자율 주행 자동차가 보다 사람처럼 주행하기 위한 전략을 수립하기 위해 필요한데, 도심 주행환경에서 여러 종류의 물체들 중 보행자를 구별해 내는 것이 무엇보다 중요한 상황을 예로 들 수 있다. 3장에서는 상기의 보행

자 인식을 위해 각 포인트 그룹의 시간에 따른 외관 변화를 학습하는 Deep Neural Network 구조가 제안되어 있다.

주행 상황에서 분류되어야 하는 대상 중, 높은 우선 순위를 갖는 또 하나의 물체로는 차량을 들 수 있다. 차량 인식 알고리즘 개발에 있어서 고려되어야 하는 중요한 문제 중 하나는, 버스, 승용차, 트럭, SUV 등 실제로 외관이 다른 여러 종류의 차량들을 종류에 관계없이 동일한 성능으로 인식해 내는 것이다. 4장에서는 상기의 문제를 해결하기 위해 차량들에서 공통으로 관찰되는 외관을 이용하는 인식 알고리즘이 제시된다. 성능 향상을 위해, 3D LIDAR 외에 단안 카메라가 추가적으로 활용되며, 상기 두 개의 센서로부터 추출된 정보는 Dempster-Shafer 이론을 이용하여 융합된다.

**주요어:** 3D LIDAR, 실시간 segmentation, Gaussian Process, 보행자 인식, Deep Neural Network, 차량 인식

**학번:** 2010-20826